Computer Vision
Assignment 12

Alireza Moradi

December 18, 2020

1 Shape Descriptors

$$Compactness = \frac{4\pi \times Area}{Perimeter^2}$$

$$Solidity = \frac{Area}{ConvexArea}$$

$$Eccentricity = \frac{MinorAxisLength}{MajorAxisLength}$$

1.1 Rhombus

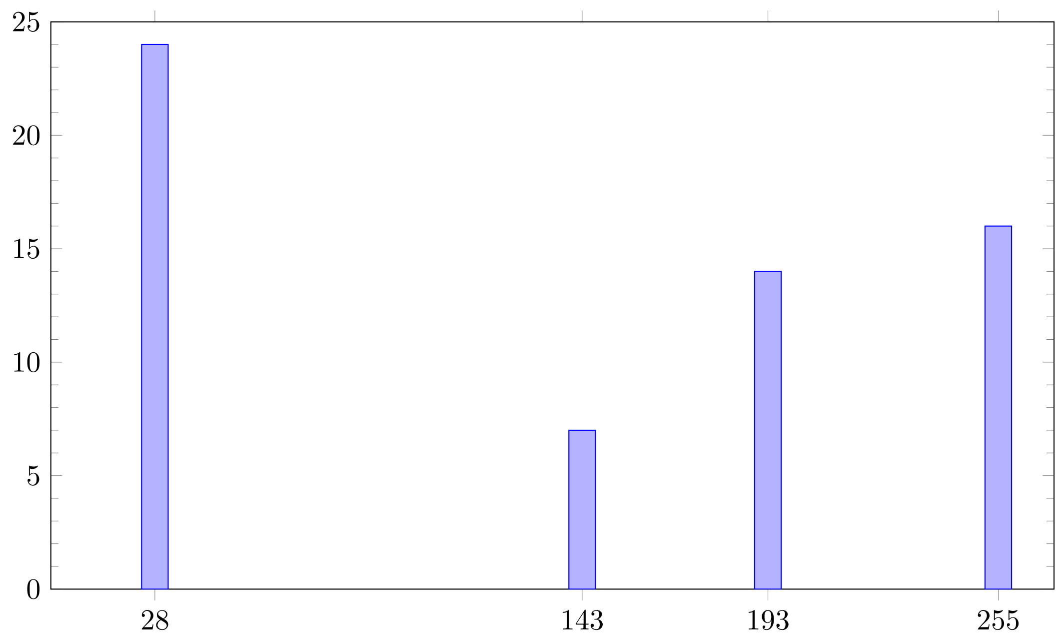
- Compactness: $\frac{4\pi \times a^2}{20a^2} = \frac{1}{5}\pi$
- Solidity: 1 (because lozenge is convex)
- Eccentricity: $\frac{a}{2a} = \frac{1}{2}$

1.2 Rectangle

- Compactness: $\frac{4\pi \times 2a^2}{36a^2} = \frac{2}{9}\pi$
- Solidity: 1 (because rectangle is convex)
- Eccentricity: $\frac{a\frac{\sqrt{5}}{2}}{a\sqrt{5}} = \frac{1}{2}$

2

$7 = (00000111)_2 \xrightarrow{Rotate90} (11000001)_2 = 193$
$62 = (00111110)_2 \xrightarrow{Rotate90} (10001111)_2 = 143$
$112 = (01110000)_2 \xrightarrow{Rotate90} (00011100)_2 = 28$
$255 = (11111111)_2 \xrightarrow{Rotate90} (11111111)_2 = 255$


25
20
15
10
5
0
28
143
193
255

3 Implementations

3.1 SVM

3.1.1 HOG

I used `skimage.feature.hog` and got results with 8 different orientations and 8×8 pixels per cell and 3×3 cells per block and this resulted in a classification with 93.7% accuracy.

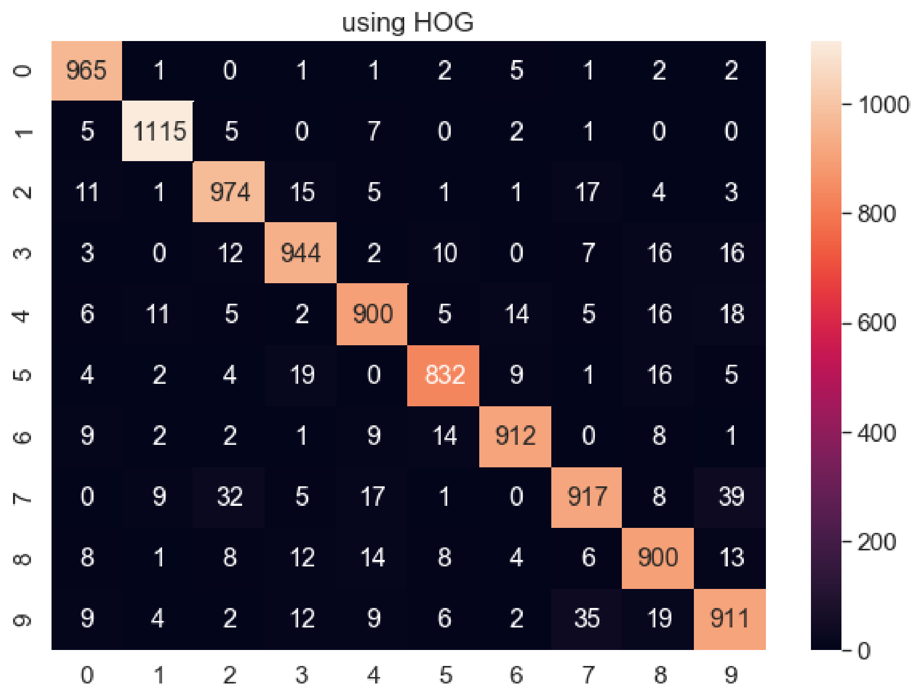


Figure 1: Confusion Matrix

```python
def classify_hog(x_train, y_train, x_test):
    '''
    Classifies images with HOG.

    Parameters:
        x_train(numpy.ndarray) : train data
        y_train(numpy.ndarray) : train labels
        x_test(numpy.ndarray) : test data

    outputs:
        predicted_labels(numpy.ndarray): labels that predicted
    '''

    x_t = x_train.copy()
    y_t = y_train.copy()
    x_te = x_test.copy()
    predicted_labels = None
```

```python

    hog_image = np.array([hog(x, orientations=8, pixels_per_cell=(8, 8),
                     cells_per_block=(3, 3), visualize=True,
                         multichannel=False)[0] for x in x_train])

    hog_image_test = np.array([hog(x, orientations=8, pixels_per_cell=(8, 8),
                     cells_per_block=(3, 3), visualize=True,
                         multichannel=False)[0] for x in x_test])
    svm = LinearSVC()
    svm.fit(hog_image,y_train)
    predicted_labels = svm.predict(hog_image_test)
    return predicted_labels
```

3.1.2 Shape Descriptors

I computed the contours with opencv and chose the largest one. Using the largest contour I computed

- `Extreme Points`: topmost, bottommost, rightmost and leftmost points of the object.
- `Aspect Ratio`: ratio of width to height of bounding rect of the object.
- `Solidity`: the ratio of contour area to its convex hull area.

of each data and used these to do the classification which resulted in 59.2% accuracy.

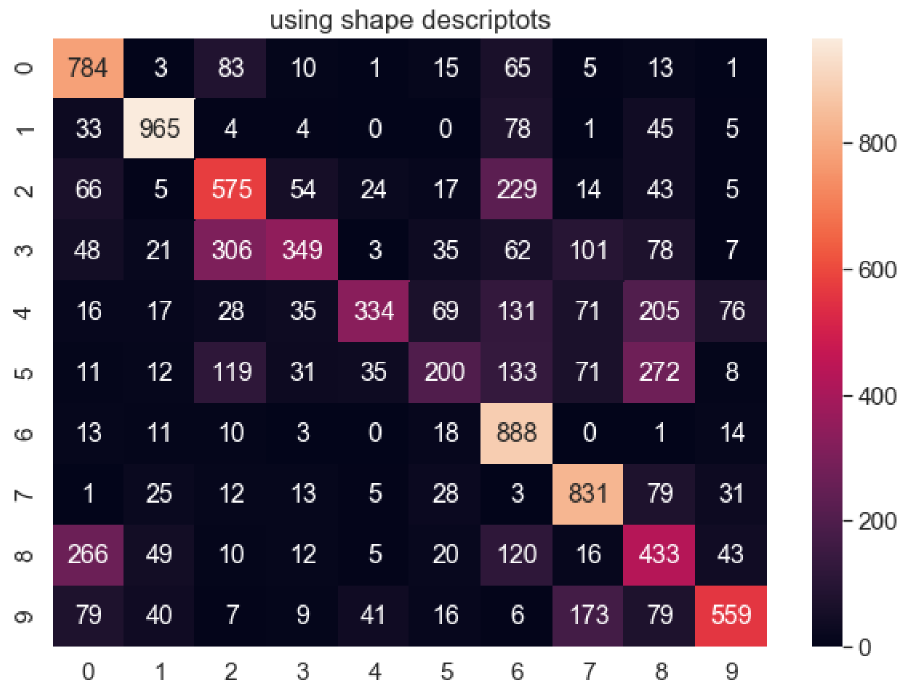


Figure 2: Confusion Matrix

```python
def extract_shape_desc(x):
    _,thresh = cv2.threshold(x,170,255,cv2.THRESH_BINARY)
    contours,hierarchy = cv2.findContours(thresh,cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
    contour_sizes = [(cv2.contourArea(contour), contour) for contour in contours]
    biggest_contour = max(contour_sizes, key=lambda x: x[0])[1]
    cnt = biggest_contour

    leftmost = np.array(cnt[cnt[:,:,0].argmin()][0]).reshape(2,1)
    rightmost = np.array(cnt[cnt[:,:,0].argmax()][0]).reshape(2,1)
    topmost = np.array(cnt[cnt[:,:,1].argmin()][0]).reshape(2,1)
    bottommost = np.array(cnt[cnt[:,:,1].argmax()][0]).reshape(2,1)

    area = cv2.contourArea(cnt)
    hull = cv2.convexHull(cnt)
```

```
    hull_area = cv2.contourArea(hull)
    solidity = float(area)/(hull_area+1e-7)

    x,y,w,h = cv2.boundingRect(cnt)
    aspect_ratio = float(w)/h

    return np.concatenate((leftmost,rightmost,
                          topmost,bottommost,[[solidity],[aspect_ratio]]),axis=0)
```

```
def classify_shape_desc(x_train, y_train, x_test):
    '''
    Classifies images by using shape descriptors.

    Parameters:
        x_train(numpy.ndarray) : train data
        y_train(numpy.ndarray) : train labels
        x_test(numpy.ndarray) : test data

    outputs:
        predicted_labels(numpy.ndarray): labels that predicted
    '''

    x_t = x_train.copy()
    y_t = y_train.copy()
    x_te = x_test.copy()
    predicted_labels = None

    data = np.squeeze([extract_shape_desc(x) for x in x_train])
    test = np.squeeze([extract_shape_desc(x) for x in x_test])

    svm = LinearSVC()
    svm.fit(data,y_train)
    predicted_labels = svm.predict(test)

    return predicted_labels
```

3.1.3 Manual LBP

I computed LBP manually (which takes a lot of time because of those nested for loops) and computed the normalized histogram of LBP with 256 bins from 0 to 255. this resulted in 70.6% accuracy.

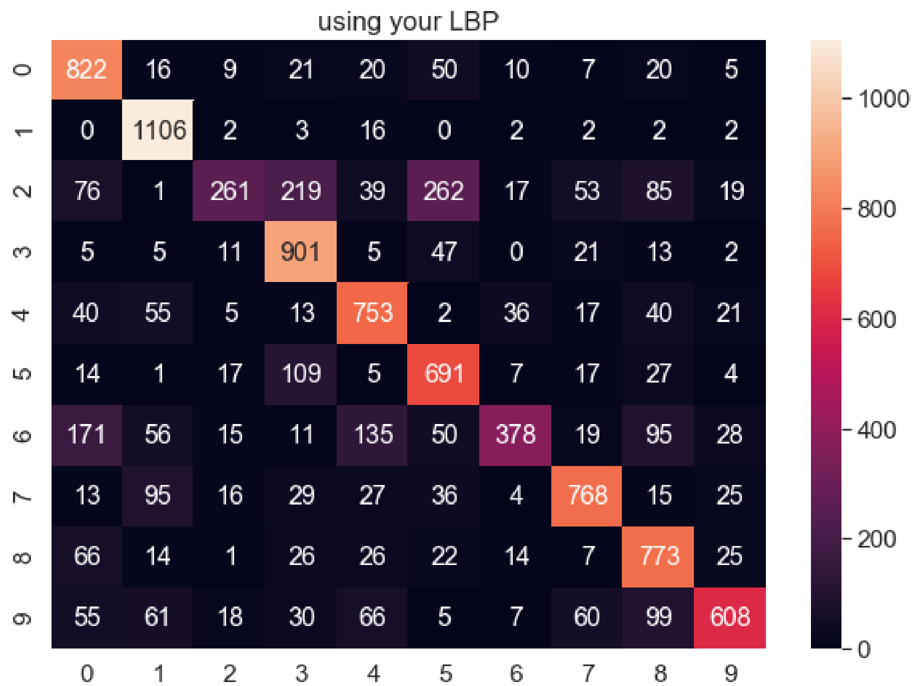


Figure 3: Confusion Matrix

Source 1 Source 2

```
def LBP(img):
    '''
    Extracts LBP features from the input image.

    Parameters:
        img(numpy.ndarray) : image data
    outputs:
        output: LBP features
    '''
    R,C = img.shape

    transformed = np.zeros((26, 26))
    for i in range (1,R-1):
        for k in range (1,C-1):

            binary_matrix = np.zeros((3, 3))
            sub_mat = img[i-1:i+2,k-1:k+2]
            mid = sub_mat[1,1]
            higher = np.where(sub_mat > mid)
```

```
            lower = np.where(sub_mat <= mid)
            binary_matrix[higher] = 1
            binary_matrix[lower] = 0
            transformed[i-1,k-1] = np.sum(np.multiply(multip_matrix,binary_matrix))

    hist, _ = np.histogram(transformed, bins=256, density=True, range=(0, 256))

    return hist
```

```
def classify_your_lbp(x_train, y_train, x_test):
    '''
    Classifies images by using your LBP.

    Parameters:
        x_train(numpy.ndarray) : train data
        y_train(numpy.ndarray) : train labels
        x_test(numpy.ndarray) : test data

    outputs:
        predicted_labels(numpy.ndarray): labels that predicted
    '''
    data = [LBP(x) for x in x_train]
    test = [LBP(x) for x in x_test]

    data = np.array(data)
    test = np.array(test)
    svm = LinearSVC()
    svm.fit(data,y_train)
    predicted_labels = svm.predict(test)

    return predicted_labels
```

3.1.4 LBP

I computed LBP by using `skimage.feature.local_binary_pattern` and computed normalized histogram. This was much faster but resulted in 57% accuracy.

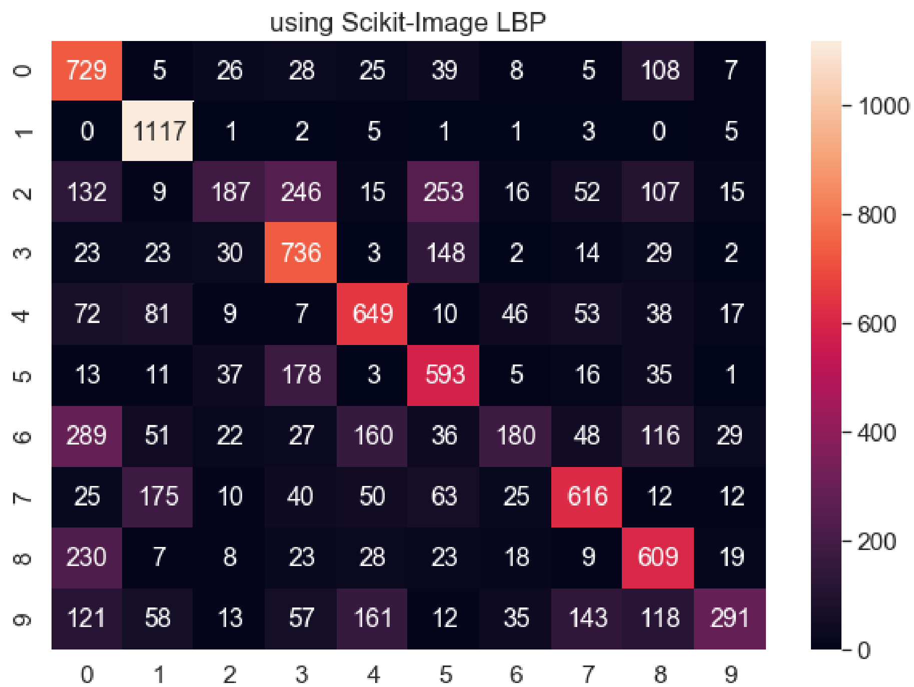


Figure 4: Confusion Matrix

```python
def classify_skimage_lbp(x_train, y_train, x_test):
    '''
    Classifies images by using Scikit-Image LBP.

    Parameters:
        x_train(numpy.ndarray) : train data
        y_train(numpy.ndarray) : train labels
        x_test(numpy.ndarray) : test data

    outputs:
        predicted_labels(numpy.ndarray): labels that predicted
    '''

    data = [np.histogram(local_binary_pattern(x,8,1), density=True
                         , bins=256, range=(0, 256))[0] for x in x_train]
    test = [np.histogram(local_binary_pattern(x,8,1), density=True
                         , bins=256, range=(0, 256))[0] for x in x_test]
    data = np.array(data)
    test = np.array(test)
    svm = LinearSVC()
    svm.fit(data,y_train)
```

```
    predicted_labels = svm.predict(test)

    return predicted_labels
```

3.2 Binary Classification

I used the following features (which are all scale invariant):

- `Extent`: the ratio of contour area to bounding rectangle area.
- `Aspect Ratio`: ratio of width to height of bounding rect of the object.
- `Solidity`: the ratio of contour area to its convex hull area.

Observing and playing with the numbers from the above features a little, I managed to classify all the leaves correctly.

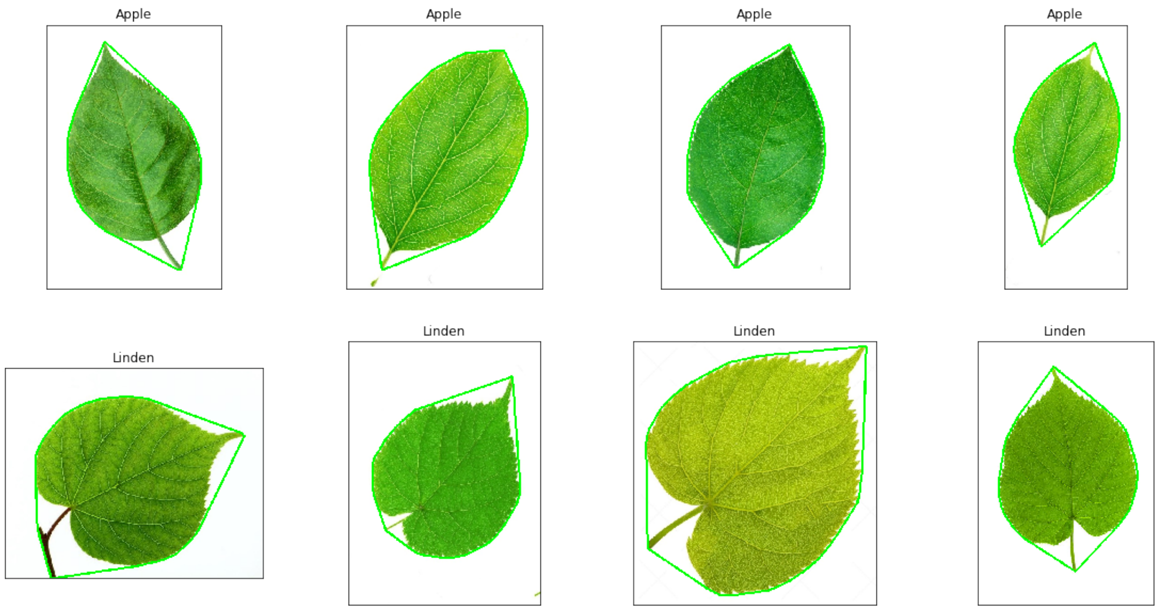


Figure 5: Classified leaves and their convex hull

```python
def classify_leaf(image):
    '''
    Classifies the input image to only two classes of leaves.

    Parameters:
        image (numpy.ndarray): The input image.

    Returns:
        int: The class of the image. 1 == apple, 0 == linden
    '''
    # apple = 0
    # linden = 1
    leaf_type = 0

    #Write your code here
    gray = cv2.cvtColor(image,cv2.COLOR_RGB2GRAY)
```

```
    ret,thresh = cv2.threshold(gray,220,255,cv2.THRESH_BINARY_INV)
    contours,hierarchy = cv2.findContours(thresh,cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

    contour_sizes = [(cv2.contourArea(contour), contour) for contour in contours]
    biggest_contour = max(contour_sizes, key=lambda x: x[0])[1]
    cnt = biggest_contour

    area = cv2.contourArea(cnt)
    hull = cv2.convexHull(cnt)
    hull_area = cv2.contourArea(hull)
    solidity = float(area)/(hull_area)

    x,y,w,h = cv2.boundingRect(cnt)
    aspect_ratio = float(w)/h

    area = cv2.contourArea(cnt)
    x,y,w,h = cv2.boundingRect(cnt)
    rect_area = w*h
    extent = float(area)/rect_area

    cv2.drawContours(image, [hull], -1, (0,255,0), 3)

    if solidity <= 0.875:
        if aspect_ratio >= 0.75:
            return 0
        else:
            if extent <= 0.6:
                return 0
            else:
                return 1
    else:
        if aspect_ratio <= 0.75:
            return 1
        else:
            if extent <= 0.6:
                return 1
            else:
                return 0

    return leaf_type
```