```
import numpy as np

import cv2

import mahotas

from skimage.feature import local_binary_pattern

from sklearn.cluster import KMeans



def extract_hu_moments(img):

    """Extract Hu Moments feature of an image. Hu Moments are shape descriptors.

    :param img: ndarray, BGR image

    :return feature: ndarray, contains 7 Hu Moments of the image

    """


    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    feature = cv2.HuMoments(cv2.moments(gray)).flatten()

    return feature



def extract_zernike_moments(img, radius=21, degree=8):

    """Extract Zernike Moments feature of an image. Zernike Moments are shapre descriptors.

    :param img: ndarray, BGR image

    :return feature: ndarray, contains 25 Zernike Moments of the image

    """


    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    feature = mahotas.features.zernike_moments(gray, radius, degree)

    return feature
```

```
def extract_haralick(img):
    """Extract Haralick features of an image. Haralick features are texture descriptors.
    :param img: ndarray, BGR image
    :return feature: ndarray, contains 13 Haralick features of the image
    """

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    feature = mahotas.features.haralick(gray).mean(axis=0)
    return feature



def extract_lbp(img, numPoints=24, radius=8):
    """Extract Local Binary Pattern histogram of an image. Local Binary Pattern features are texture
descriptors.
    :param img: ndarray, BGR image
    :return feature: ndarray, contains (numPoints+2) Local Binary Pattern histogram of the image
    """
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    lbp = local_binary_pattern(gray, numPoints, radius, method='uniform')
    n_bins = int(lbp.max() + 1)
    feature, _ = np.histogram(lbp.ravel(), bins=n_bins, range=(0, n_bins), density=True)
    return feature



def extract_color_histogram(img, n_bins=8):
    """Extract Color histogram of an image.
    :param img: ndarray, BGR image
    :return feature: ndarray, contains n_bins*n_bins*n_bins HSV histogram features of the image
    """
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV) # convert the image to HSV color-space
    hist  = cv2.calcHist([hsv], [0, 1, 2], None, [n_bins, n_bins, n_bins], [0, 180, 0, 256, 0, 256])
    cv2.normalize(hist, hist)
    feature = hist.flatten()
    return feature



def extract_global_features(img):
    """Extract global features (shape, texture and color features) of an image.
    :param img: ndarray, BGR image
    :return global_feature: ndarray, contains shape, texture and color features of the image
    """

    hu_moments = extract_hu_moments(img)
    zernike_moments = extract_zernike_moments(img)
    haralick   = extract_haralick(img)
    lbp_histogram  = extract_lbp(img)
    color_histogram  = extract_color_histogram(img)
    global_feature = np.hstack([hu_moments, zernike_moments, haralick, lbp_histogram,
color_histogram])

    return global_feature



def extract_keypoints(keypoint_detector, image):
    keypoints, descriptors = keypoint_detector.detectAndCompute(image, None)
    return [keypoints, descriptors]
```

```
def flatten_keypoint_descriptors(X_train_local_features):
    descriptor_list_train = np.array(X_train_local_features[0])
    for remaining in X_train_local_features[1:]:
            descriptor_list_train = np.vstack((descriptor_list_train, remaining))
    return descriptor_list_train


def cluster_local_features(descriptor_list_train, n_clusters=20):
    kmeans = KMeans(n_clusters=n_clusters)
    kmeans.fit(descriptor_list_train)
    return kmeans


def extract_local_features(X_train_local_features, X_test_local_features, n_clusters=20):
    # flatten keypoint_descriptors
    descriptor_list_train = flatten_keypoint_descriptors(X_train_local_features)
    descriptor_list_test = flatten_keypoint_descriptors(X_test_local_features)

    # cluster keypoint descriptors
    kmeans = cluster_local_features(descriptor_list_train, n_clusters=n_clusters)
    descriptor_clustered_train = kmeans.predict(descriptor_list_train)
    descriptor_clustered_test = kmeans.predict(descriptor_list_test)

    # For each image, count number of keypoints in each cluster that the image has
    X_clustered_train = np.array([np.zeros(n_clusters) for i in range(len(X_train_local_features))])
    old_count = 0
    for i in range(len(X_train_local_features)):
            nb_descriptors = len(X_train_local_features[i])
```

```
        for j in range(nb_descriptors):
            idx = descriptor_clustered_train[old_count+j]
            X_clustered_train[i][idx] += 1
        old_count += nb_descriptors

X_clustered_test = np.array([np.zeros(n_clusters) for i in range(len(X_test_local_features))])
old_count = 0
for i in range(len(X_test_local_features)):
    nb_descriptors = len(X_test_local_features[i])
    for j in range(nb_descriptors):
        idx = descriptor_clustered_test[old_count+j]
        X_clustered_test[i][idx] += 1
    old_count += nb_descriptors

return X_clustered_train, X_clustered_test
```