# Tutorial:
# Fuzzy Computer Vision Toolbox

FUZZ 2017


## Abstract

This is a handbook for beginner to understand the general pipeline and algorithms in computer vision. A guideline to build yourself the first computer vision application with computer vision and Fuzzy toolbox.



Chern Hong Lim
chlim@acd.tarc.edu.my / lch8743@hotmail.com

Chee Seng Chan
cs.chan@um.edu.my

# Table of Contents

# Pre-requisite

There are a few prerequisites before you start this practical, please ensure you have installed the following toolboxes or libraries in your computer. You may follow the steps below for installation:

1. Install Python (Recommend Anaconda Python 2.7 version)
    i. Download from: https://www.continuum.io/downloads

2. Install opencv library (version 2.4.x.x)
    i. Download opencv library from: http://opencv.org/releases.html
    ii. Double-click to extract the opencv.
    iii. Go to "opencv/build/python/2.7/x64 folder."
    iv. Copy cv2.pyd to your python directory in the "lib/site-packages".

3. Install scikit-image package.
    i. Open anaconda prompt
    ii. Type "pip install scikit-image"
    iii. Web reference: http://scikit-image.org/

4. Install scikit-learn package.
    i. Open anaconda prompt
    ii. Type "pip install scikit-learn"
    iii. Web reference: http://scikit-learn.org/stable/

# Introduction

Computer vision is an area in computer science that endow the capability to extract, analyse, and understand a single image or sequence of images to a computer. It involves the development of theories and algorithms to transform visual images into descriptor(s) which is useful for visual context understanding, recognition task, and decision making. This is an emerging domain due to vast spectrum of applications, for examples, scene understanding, object recognition, video tracking, event detection, and many others. To achieve this, it is important to understand the general pipeline of a computer vision algorithm as illustrated in Figure 1 and this pipeline is used throughout this tutorial.

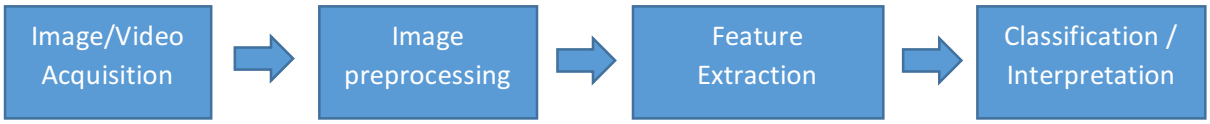



Figure 1: General pipeline of computer vision algorithm

In this practical, the participant will be instructed on how to perform each of the steps mentioned above with supported by a few notable python libraries. In conjunction, a new toolbox named Fuzzy Qualitative Rank Classifier (FQRC) will be introduced for Fuzzy Classification. FQRC ***[Lim et. al., 2014]*** is proposed to overcome the limitations of Crisp classification and ordinary Fuzzy Inference System (FIS) in computer vision task. Both Crisp and Fuzzy techniques will be introduced in the tutorial with an application on image classification. The topics included in the tutorial are as follows:

A. Image / Video Acquisition
    a. Read Source
B. Image Preprocessing
    a. Resize image
    b. Image conversion
    c. Morphological operation
    d. Image filtering
C. Feature Extraction
    a. Color detection
    b. Edge detection
    c. Corner detection
    d. Keypoint Detection
    e. LBP
    f. Feature Representation
D. Image Classification
    a. Crisp Classification
    b. Fuzzy Classification

Please note that the computer vision functions used in this toolbox is built on top of the existing well-known libraries in Python which are Opencv, scikit-image, and scikit-learn library except the FQRC toolbox. For simplicity, the necessary functions are packaged in a main python library namely FCVT. The overview of our toolbox is illustrated in Figure. 1.

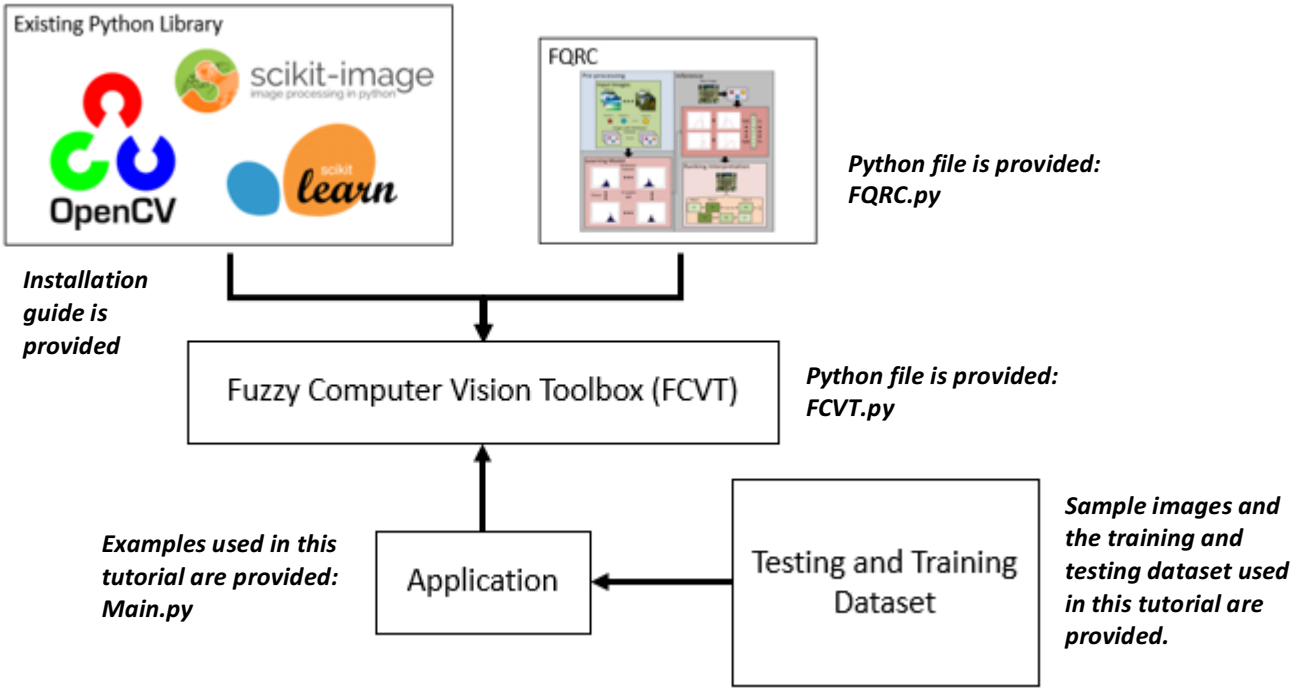



Figure 1: Overview of FCVT

User can import FCVT to access all the functions in the toolbox and each of them will be discussed in this tutorial.

```
import FCVT as fcvt
```

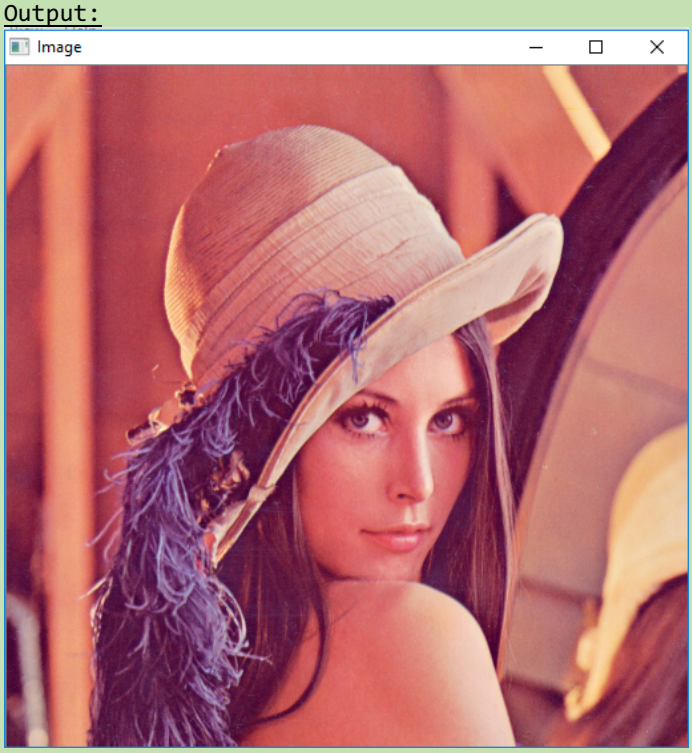
# Section A: Image Acquisition

## (1) Read Source

For a computer vision application, the process begin with acquiring image or video as the input for further processing. By using FCVT, this step can be done by calling the "`IA_readSource`" function with the directory is provided in full or merely filename (if the file is in the local folder). The user can indicate "`display =True`" to visualize the image.

```
IA_readSource( sourceDir, display )

Sample Code:
sourceDir = 'Lenna.png'
image = fcvt.IA_readSource(sourceDir, display=True)
```

Output:



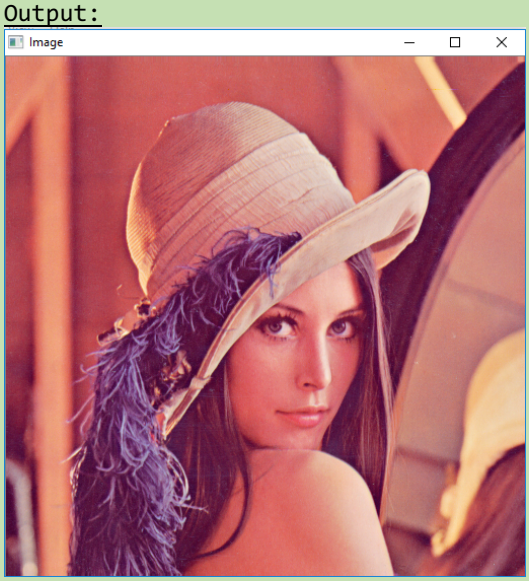
# Section B: Image Pre-Processing

## (1): Image resize

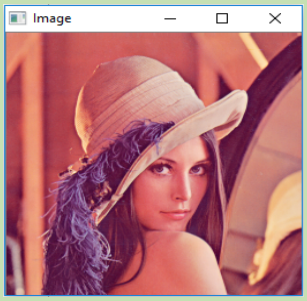
An image can be resized with the "IP_resize" function where "sx" and "sy" indicate the scale that the image to be resized (Example: 0.5 means to downsize the image into ½ of the original size).

```
IP_resize(image, sx, sy, display)

Sample Code:
imageResize = fcvt.IP_resize(image, 0.5, 0.5, True)
```

Output:




(Original Image)




(Resized Image)

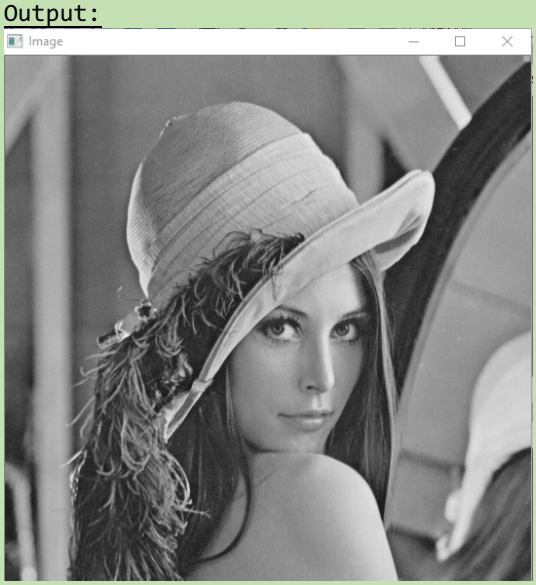
## (2): Image conversion

There are sometimes an image is preferable to be converted into different color space to simplify or to support the later processes such as feature extraction. With FCVT, user can easily convert to grayscale or binary image from color image with the following functions.

* Note: The threshold for the binary conversion is using Otsu method ***[Otsu, 1979]***.

```
IP_convertGray(image, display)
IP_convertBinary(image, display)

Sample Code:
imageGray = fcvt.IP_convertGray(image, True)
imageBinary = fcvt.IP_convertBinary(image, True)
```

Output:





## (3): Image Morphological Operations

Images may contain numerous imperfections. In particular, the binary regions produced by simple thresholding are distorted by noise and texture (example, gaps between the pixels). Morphological image processing pursues the goals of removing these imperfections by accounting for the form and structure of the image. Morphological operators often take a binary image and a structuring element as input and combine them using a set operator (intersection, union, inclusion, complement). They process objects in the input image based on characteristics of its shape, which are encoded in the structuring element. There are four common types of morphological operations which are; erosion, dilation, opening and closing.

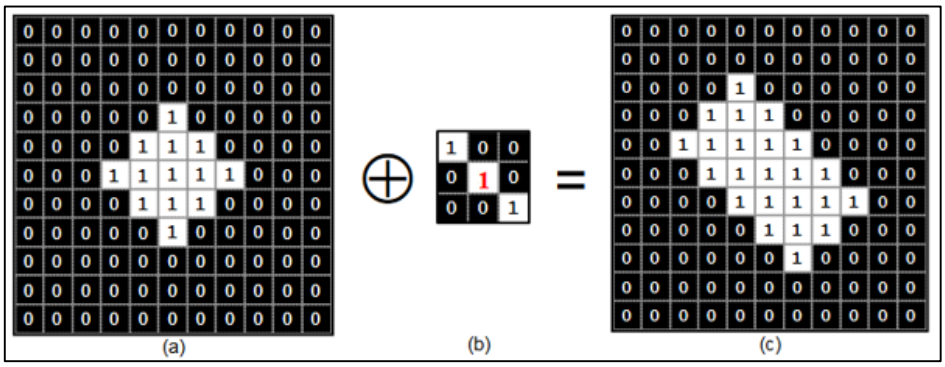



Figure 2: Dilation with structuring element as (b)

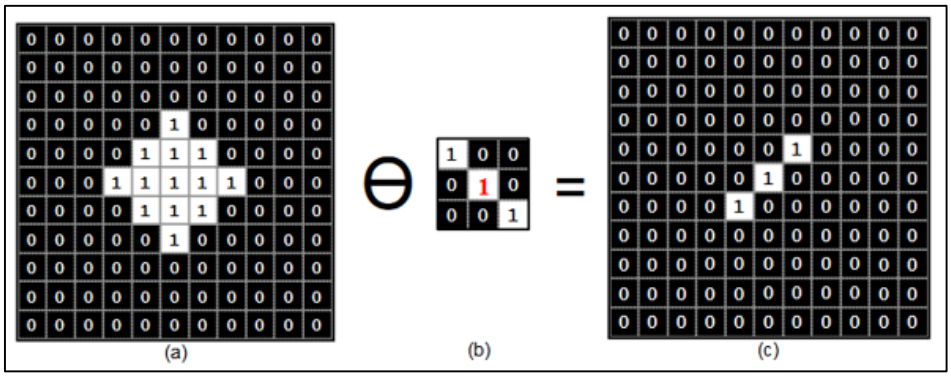



Figure 3: Erosion with structuring element as (b)

Following lines perform morphological operation with a Digit image.

*Note: The method argument can be the followings:

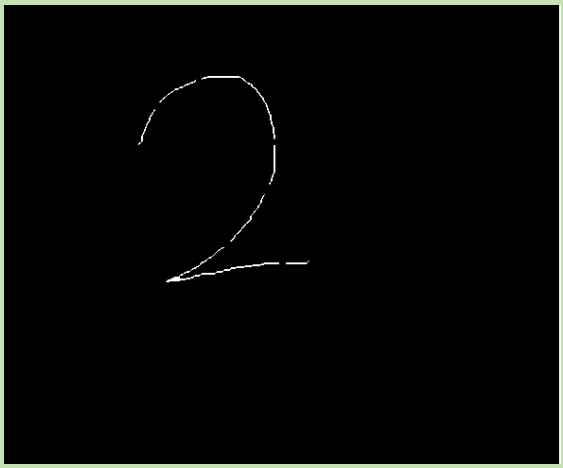
i. erosion
ii. dilation
iii. open (Erode then dilate)
iv. closing (Dilate then erode)

And the kernel is the size of kernel specify in python tuple, example, (3,3) for 3x3 kernel.

```
IP_imageMorph(image, method, kernelSize, display)

Sample Code:
imageDigit = fcvt.IA_readSource('Digit3.png', True)
imageMorph = fcvt.IP_imageMorph(imageDigit, 'erosion', (3,3), True)
imageMorph = fcvt.IP_imageMorph(imageDigit, 'dilation', (5,5), True)
imageMorph = fcvt.IP_imageMorph(imageDigit, 'opening', (3,3), True)
imageMorph = fcvt.IP_imageMorph(imageDigit, 'closing', (5,5), True)
```

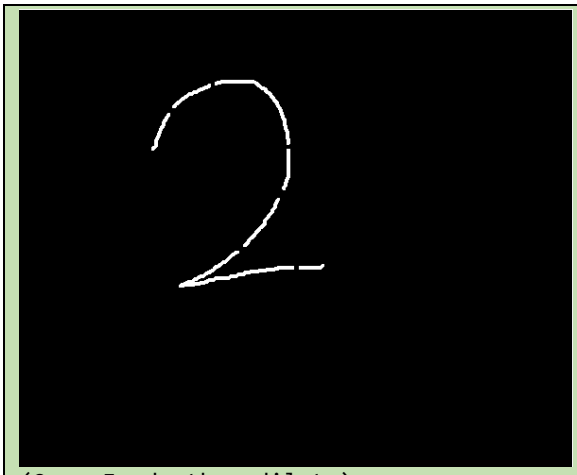
Output:

(Original Image)



(Erosion – The noise are eliminated)

(Dilation – The gap are connected)



(Open-Erode then dilate)

(Close-Dilate then erode)

## (4): Image Filtering

Image noise is random (not present in the object image) variation of brightness or color information in images. It is an unwanted signal that could be an obstacle in the later processes (e.g. feature extraction) and it might affect the overall system performance. Thus, it is important to have noise removal in the image preprocessing step. Following lines demonstrates noise removal by using three different filters (Average, Gaussian, and Median) on camera man image with salt and pepper noise.

*Note: The method argument can be the followings:

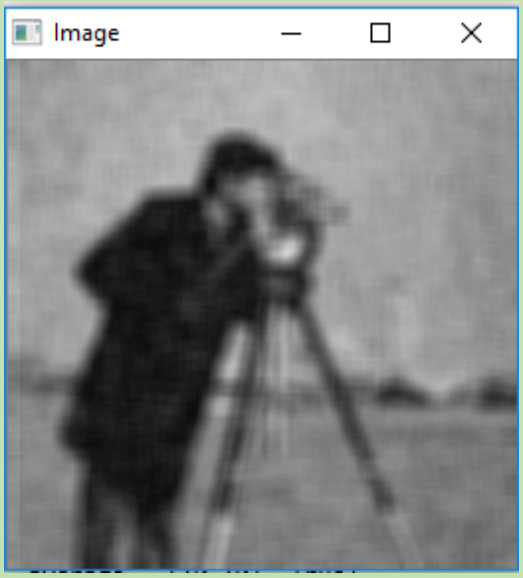
i. average
ii. Gaussian
iii. median

And the kernel is the size of kernel specify in tuple, example, (3,3) for 3x3 kernel.

```
IP_imageFilt(image, method, kernel, display)

Sample Code:
imageCameraman = fcvt.IA_readSource('cameraman_noise.jpg', True)
imageFiltered = fcvt.IP_imageFilt(imageCameraman, 'average', (10,10), True)
imageFiltered = fcvt.IP_imageFilt(imageCameraman, 'gaussian', (5,5), True)
imageFiltered = fcvt.IP_imageFilt(imageCameraman, 'median', (3,3), True)
```

Output:


(Original image with Salt and pepper noise)




(Result from Average filter)


(Result from Gaussian filter)

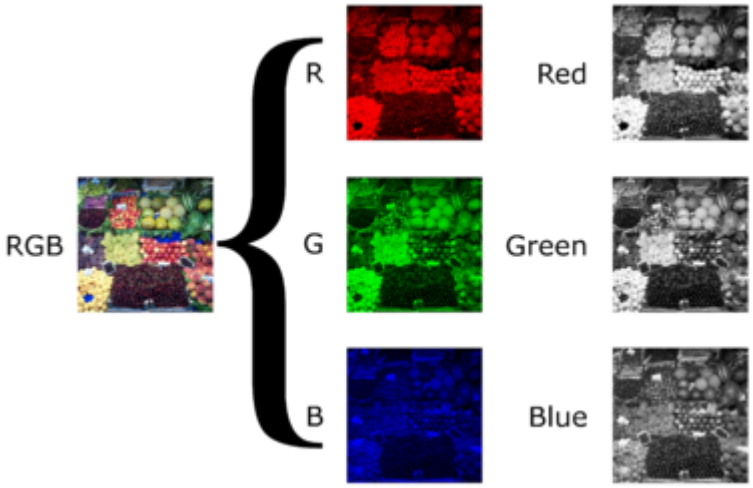

(Result from median filter)

# Section C: Feature Extraction

## (1): Visual Features

### (a): Color Detector

A digital image is normally represented using RGB color space in three-dimensional matrix with each dimension representing the intensity value of Red, Green, and Blue respectively. The intensity is normally from 0 to 255.



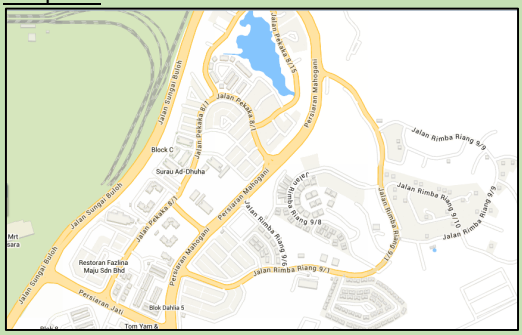

The intensity value for each pixel in the R, G ,B layer is normally from 0 to 255.

Figure 4: Components of color image

Color is very useful visual features that commonly used to segment a region of interest from an image. Following provide an example to segment different regions from the map image using color feature and visualize it in different windows such as lake, road, field, and housing area. This can be easily done by distinguish each of them using the different range of RGB values. You may check the RGB values for the map using the following website: http://imagecolorpicker.com/. The lowerbound and upperbound in the example indicate the python list with [R, G, B] value.

```
FE_colorDetection(image, lowerbound, upperbound, display)

Sample Code:
imageMap = fcvt.IA_readSource('map.png', True)
imageColor = fcvt.FE_colorDetection(imageMap, lowerbound, upperbound, True)
```

Output:



lowerbound = [120,185,240]
upperbound = [150,205,260]

lowerbound = [245,205,100]
upperbound = [265,230,170]

lowerbound = [204,220,179]
upperbound = [224,240,199]

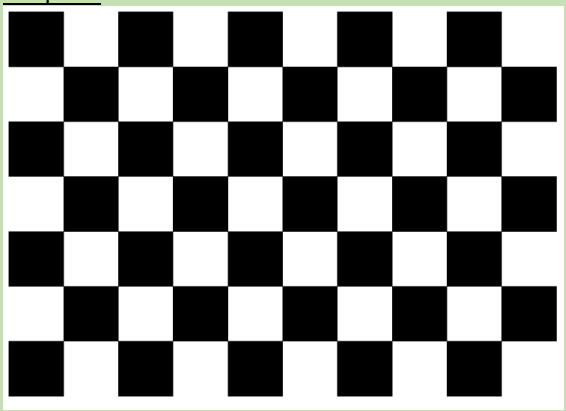
lowerbound = [235,214,228]
upperbound = [255,254,248]

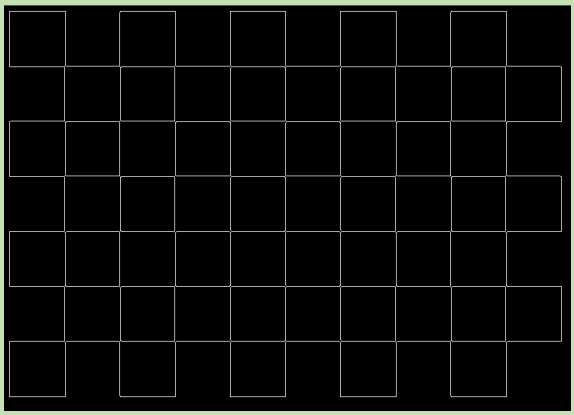
## (b): Edge Detector

Edge of an image is detected by the capturing the sharp changes in the image brightness. It provides the contour information or an outline of an object. Depends on the application, by applying an edge detection algorithm to an image may significantly reduce the amount of data to be processed in the next processing stage. It can therefore filter out information that may be regarded as less relevant, while preserving the important structural properties of an image. Following example used canny operator ***[Canny, 1986]*** to extract the edge of the Chessboard image.

```
FE_edgeDetection(image, method, display)

Sample Code:
imageChessboard = fcvt.IA_readSource('imageChessboard.png', True)
imageEdge = fcvt.FE_edgeDetection(image, True)
```

Output:



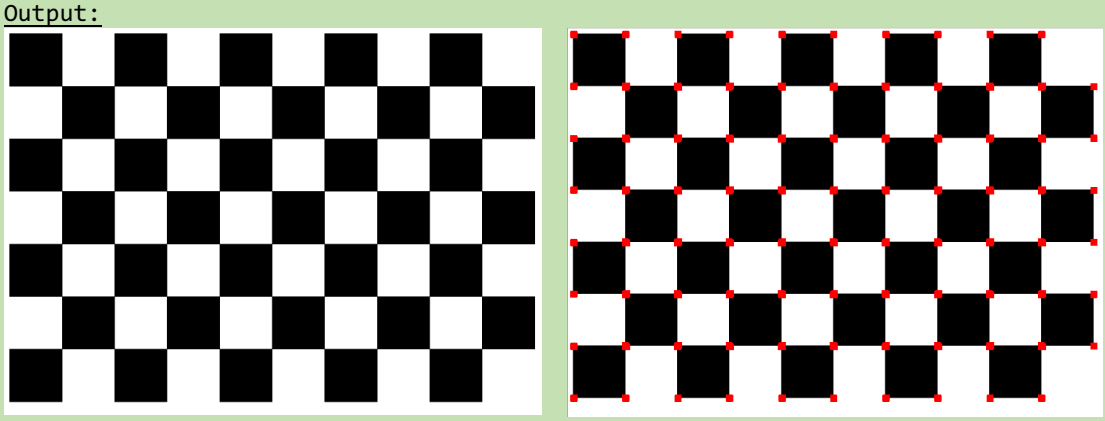


### (c): Corner detector

Same as color and edge, corner is another visual feature that provide structure information of an image. In FCVT, Harris corner detector ***[Harris & Stephens, 1988]*** is used to extract the corner of a given image. An example is provided below showing the effectiveness of the corner detector in extracting all the corner from the chessboard image.

```
FE_cornerDetection(image, display)

Sample Code:
imageChessboard = fcvt.IA_readSource('imageChessboard.png', True)
imageCorner = fcvt.FE_cornerDetection(image, True)
```

Output:



* The corners are indicate by a red dot.

## (2): Holistic Image Features

### (a): Local Binary Pattern detector

Local Binary Pattern (LBP) ***[He & Wang, 1990]*** is an image descriptor generated for classification in computer vision. It is widely used in two-dimensional texture analysis. LBP is simple yet efficient texture operator which labels the pixels of an image by thresholding the neighborhood of each pixel and considers the result as a binary number as shown in Figure 5. It is also robust to illumination change and its simplicity makes it possible to analyze images in challenging real-time settings. Generally, block approach is employed to obtain the local LBP descriptor for every local block and concatenate together to generate a descriptor to describe or represent the image. The algorithm that extract one descriptor to describe a whole image is called a holistic image feature.

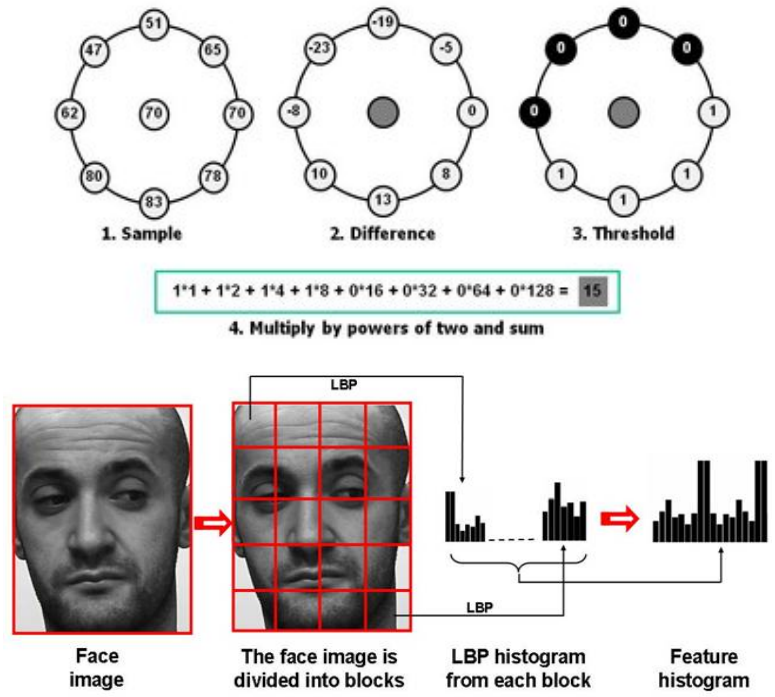



Figure 5: LBP feature extraction (***Image adopted from Wikipedia***)

```
FE_LBPDetection(image, method, display)

Sample Code:
imageLBP = fcvt.FE_LBPDetection(image, 'uniform', True)
```

```
Output:

imageLBP = [0.0467 0.0318 0.0178 0.0116 0.0085 0.0074 0.0070 0.0080 0.0095 0.0128
0.0201 0.0408 0.0603 0.0443 0.0212 0.0142 0.0096 0.0083 0.0075 0.0079 0.0097 0.0131
0.0191 0.0298 0.0522 0.4810]

* The histogram of the LBP feature.
```

## (3): Local Image Features

### (a): Keypoint detector

Unlike LBP, keypoint detector generates a collection of feature descriptor that describe the interest point towards the object(s) in an image. The descriptors extracted from a training image, can then be used to identify the object when attempting to locate the object in a test image containing many other objects.

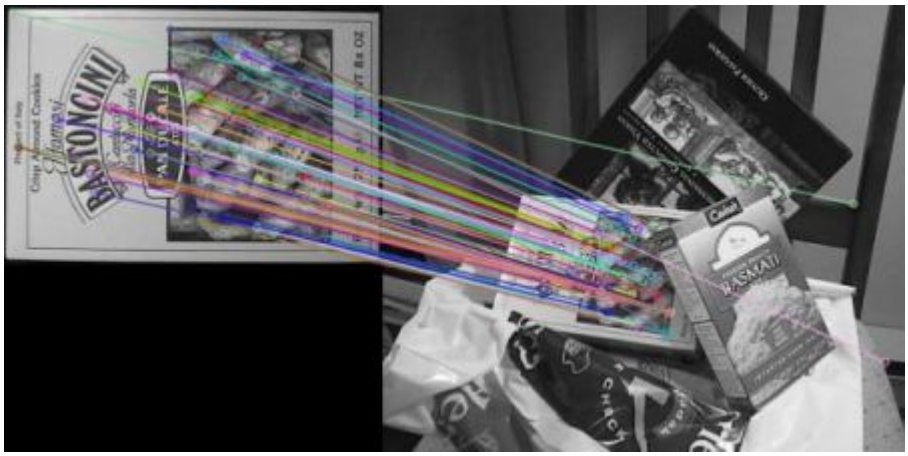


Figure 6: Image recognition with keypoint ***(Image adopted from google image)***

To perform reliable recognition, it is important that the features extracted from the training image be detectable even under changes in image scale, orientation and illumination. Such points usually lie on high-contrast regions of the image, such as object edges. The extracted keypoints do not restrict on the location in an image which means that the relative positions between them in the original scene and testing image shouldn't change from one image to another. Scale Invariant Feature Transform (SIFT) ***[Lowe, 2004]***, and Speeded up robust features (SURF) ***[Bay et al., 2008]*** are two notable keypoints detector in computer vision. SIFT uses Scale-space extrema detection with the aid of Difference of Gaussian (DoG) and Orientation assignment to achieve the robustness in scale and orientation invariant (refer to Figure 7). On the other hand, SURF extract the keypoint by approximation of the determinant of Hessian blob detector with a precomputed integral image. SURF is claimed to faster than SIFT in the processing.

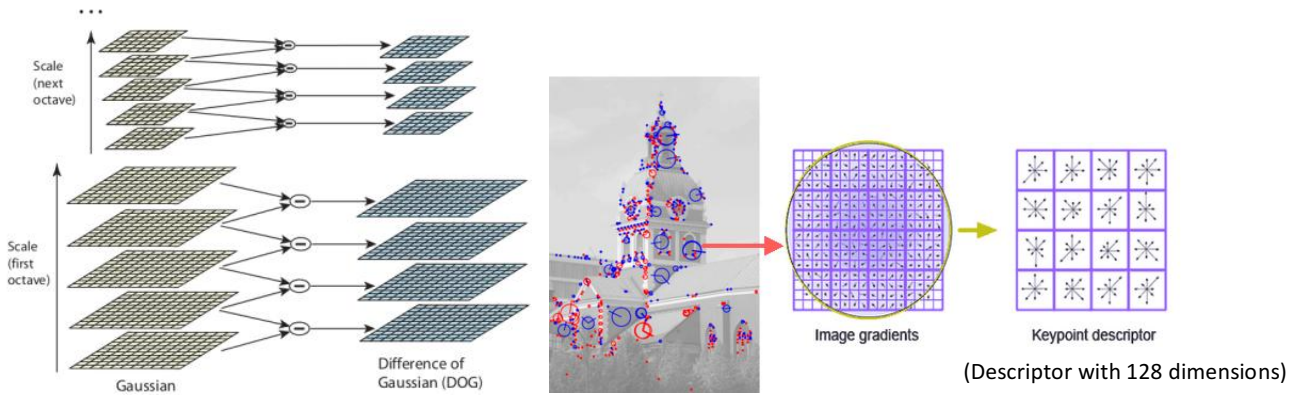


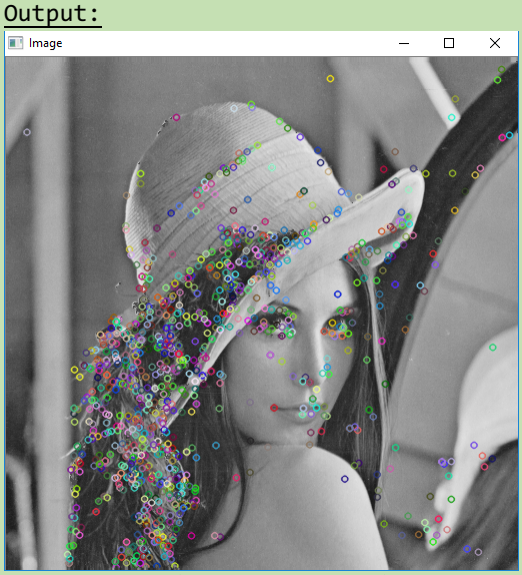

Figure 7: SIFT feature extraction ***(Image adopted from google image)***

*Note: The method argument can be the followings:

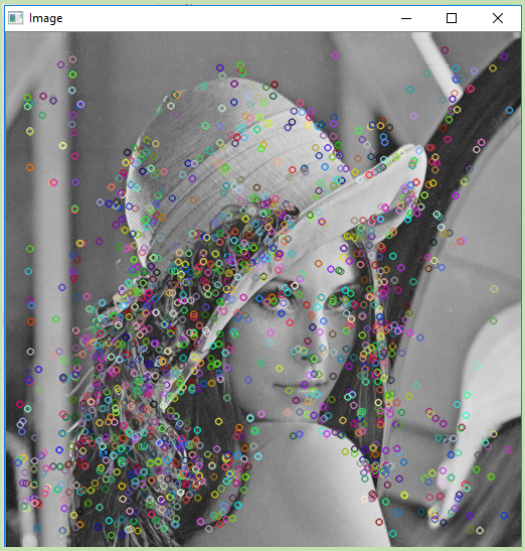
i. SIFT
ii. SURF

```
FE_keypointDetection(image, method, display)

Sample Code:
imageKeyPoint = fcvt.FE_keypointDetection(image, 'SIFT', True)
imageKeyPoint = fcvt.FE_keypointDetection(image, 'SURF', True)
```

Output:




(SIFT keypoints)




(SURF keypoints)

## (4): Feature Representation using Bag of Feature (For keypoints detector)

In computer vision, a descriptor (feature vector) is used to represent an image or a video event. The descriptors of images from a dataset will be used to train a classifier with Supervised Learning approach (e.g. Support Vector Machine) to generate a classifier (so called trained model) that can be applied later in image classification task. Holistic feature like LBP provides us a descriptor that describe the whole image but local features such as keypoints detector generates numerous descriptors from one image. In image classification, only one descriptor per image will feed for training and testing purpose. This makes local features having difficulty to directly used in classification step. In this tutorial, Bag of Feature (BoF) ***[Fei-Fei & Perona, 2005]*** approach is introduced as the solution.

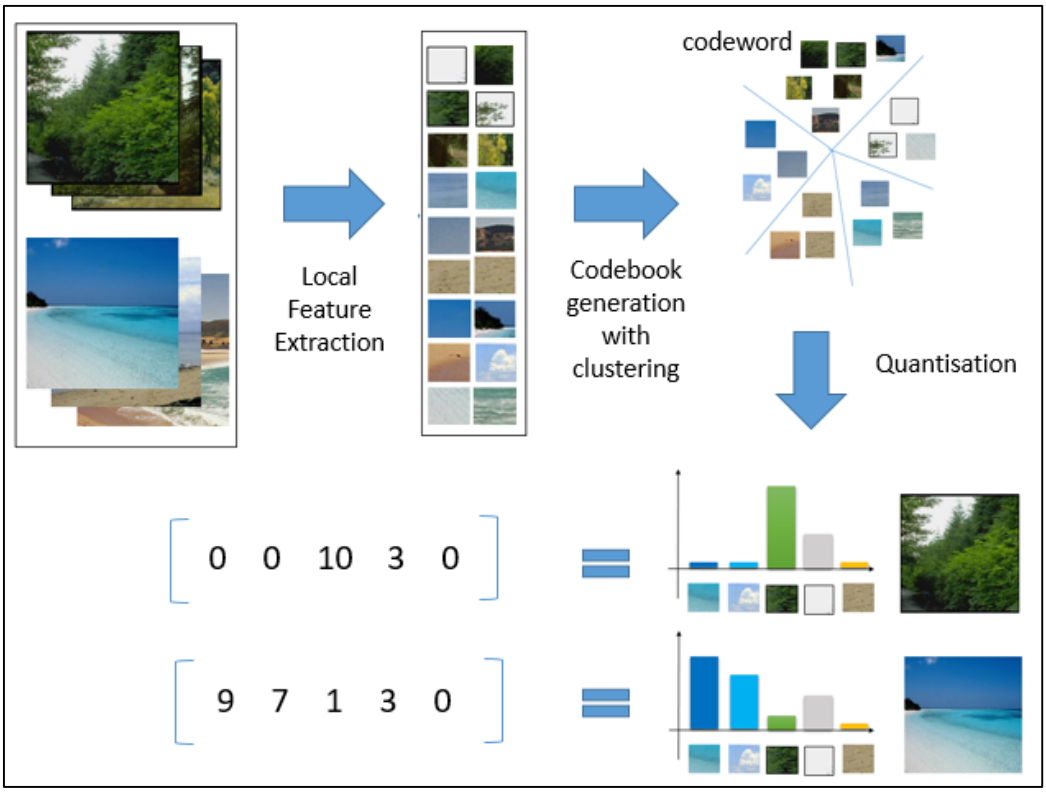



Figure 8: Overall BoF framework.

BoF origins from Bag of Words ***[Salton & McGill, 1983]*** with the objective to classify a document by computing the frequency of words in dictionary. This approach has proven to works well in image classification ***[Fei-Fei & Perona, 2005]***. The overall process of BoF is shown in Figure 8.

The local features (e.g. keypoint) are first assigned to respective cluster by using Unsupervised Learning approach (e.g. kmeans clustering) and each cluster is called a Codeword. The Codewords form a dictionary call codebook. The descriptor of an image is form by computing the frequency of codewords that are available in the codebook. This step is called Quantisation. In the later stage, the descriptors generated from the dataset can be used to train the classifier for recognition purpose. In general, image from the same class will have similar composition of the codewords. Below are the two functions to generate codebook once the local features are extracted:

```
FE_Clustering(feaMat, clusterNo)
FE_Quantisation(imageKeyPoint, cluster)

Sample Code:
cluster = fcvt.FE_Clustering(imageKeyPoint , 5)
descriptor = fcvt.FE_Quantisation(imageKeyPoint, cluster)
```

```
Output:
Descriptor = [34 65 78 23 12]

* The descriptor is the frequency of the codewords appear in the respective image.
```

## Section D: Image Classification

Image Classification is a task of assigning an input image with label from a fixed set of categories. First, a sufficient amount of training images from different classes (E.g. Figure 9) are used to train a classifier and the classifier will be used to classify a testing image into respective class. OSR scene dataset ***[Oliva & Torralba, 2001]*** is used in this tutorial and is available from http://people.csail.mit.edu/torralba/code/spatialenvelope/.

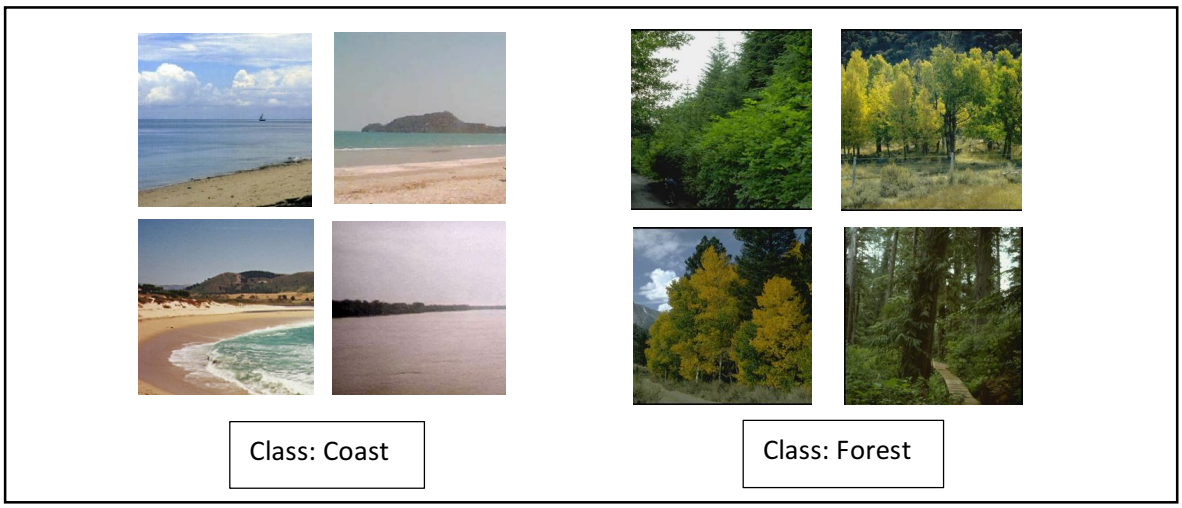



Figure 9: Training Images

**(a) Coast** **(b) ?** **(c) Forest**

Figure 10: Testing Images

The conventional goal of the classification tasks is to assign an unknown scene image to one of the several possible classes. For example, Fig. 10(a) is a Coast class while Fig. 10(c) is a Forest class. Intentionally, most state-of-the-art approaches in image classification are exemplar-based and due the images used are mutually exclusive to each other. However, this has over simplifies the complex real world problem to a simple Crisp classification task. As a result of this, classification errors often occur when the image classes that are overlap in the selected feature space. For example, it is unclear that in Fig. 10(b) is a Coast class or a Forest class.

In many cases, complex real world images are non-mutually exclusive where different people are likely to respond inconsistently. For examples, scene understanding, human motion analysis, emotion recognition, etc. Inspired by the fuzzy set theory proposed by Lotfi Zadeh ***[Zadeh, 1965]***, this work study the effectiveness of using Fuzzy Qualitative Rank Classifier (FQRC) ***[Lim et al., 2014]*** to relax the assumption that the aforementioned cases are mutually exclusive. Therefore, these images can be somewhat arbitrary and possibly sub-optimal.

This tutorial covers both Crisp classification and FQRC classification. In FCVT toolbox, the image classification task can be done by simply invoke the following function.

```
Image_classification(Train_folder, Test_folder, feature, method)
```

User is only require to provide the directory of the train folder, test folder, type of feature ('SIFT', 'SURF', or 'LBP'), and classification method ('Crisp' or 'Fuzzy'). The user may put as many classes in the training folder and testing folder with additional folder named "classX" for training and testing purpose. No limit on the amount of testing and training data but the processing time and visualization might be affected.

## (1) Crisp Classification

### (a) SVM Classification

Support vector machine (SVM) is a supervised learning model. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other.

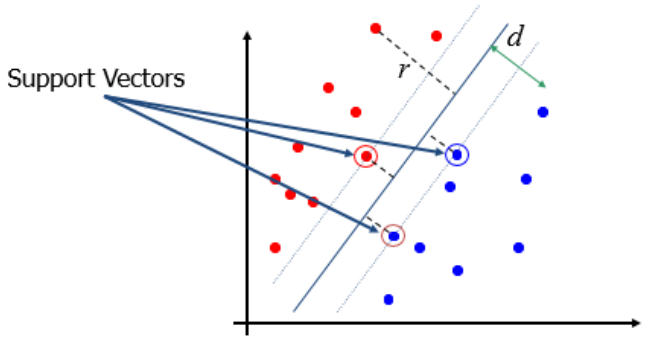



Figure 11: Illustration of SVM

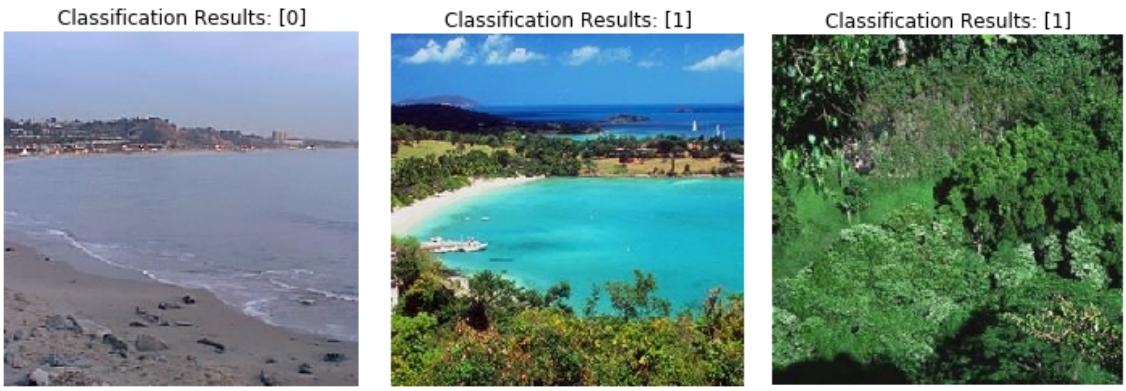
### (b) Image Classification with Crisp Approach

SVM is applied in the Crisp classification where the output of the classification results are in crisp whereby here is determined by the index of the class in the training folder. For example, if the system is trained with two classes as input from the training folder, the output will be either [0] to indicate the image that belongs to first category, and [1] to the second category. Subsequent numbers will be used if the class in the training folder is more than two.

Sample code:

```
Output_crisp = fcvt.Image_Classification('Training', 'Testing', 'SIFT', 'Crisp')
```

Output:




## (2) Fuzzy Classification (FQRC) *[Lim et al., 2014]*

It is undeniable that crisp classification approach works well in many classification tasks but limited to the images or objects that are mutually exclusive. However, Ambiguity or uncertainty is a pervasive element of many real-world decision making processes. Variation in decisions is a norm in this situation when the same problem is posed to different subjects. Computer vision tasks such as scene understanding, emotion detection, and human motion analysis are fall into this category. Unfortunately, crisp classifier is unable to deal with the uncertainty or ambiguous cases effectively as the classifier will assign one and only label to each testing image.

As a solution, Fuzzy approach is applied to relax the above issue. Fuzzy set theory ***[Zadeh, 1965]*** is widely used in control system nowadays with incorporation of Fuzzy Inference System (FIS). But, in the nature of computer vision task, it is likely to deal with high dimensional feature input. This cause some issues to directly apply FIS in the work. The issues are:

i. Tedious job in manually determine the membership function for each feature.
ii. Almost impossible to produce the associate rules for high dimension feature space.
iii. Complex in inference based on extensive rules.

With this, FQRC is proposed with the strength in capable of performing feature learning to automate the fuzzy membership generation, and the inference is not governed by any rules. At the end of the process, FQRC is capable of generate the answer in terms of the ranking result annotate different confident values. Such interpretation is closer how human perform reasoning to the ambiguous cases. FQRC consists of four stages: 1) Pre-processing (feature extraction); 2) Learning model (fuzzy membership generation); 3) Inference and 4) Ranking interpretation as illustrated in Figure 12. Interested reader can download the paper from http://cs-chan.com/doc/TFS2014.pdf.

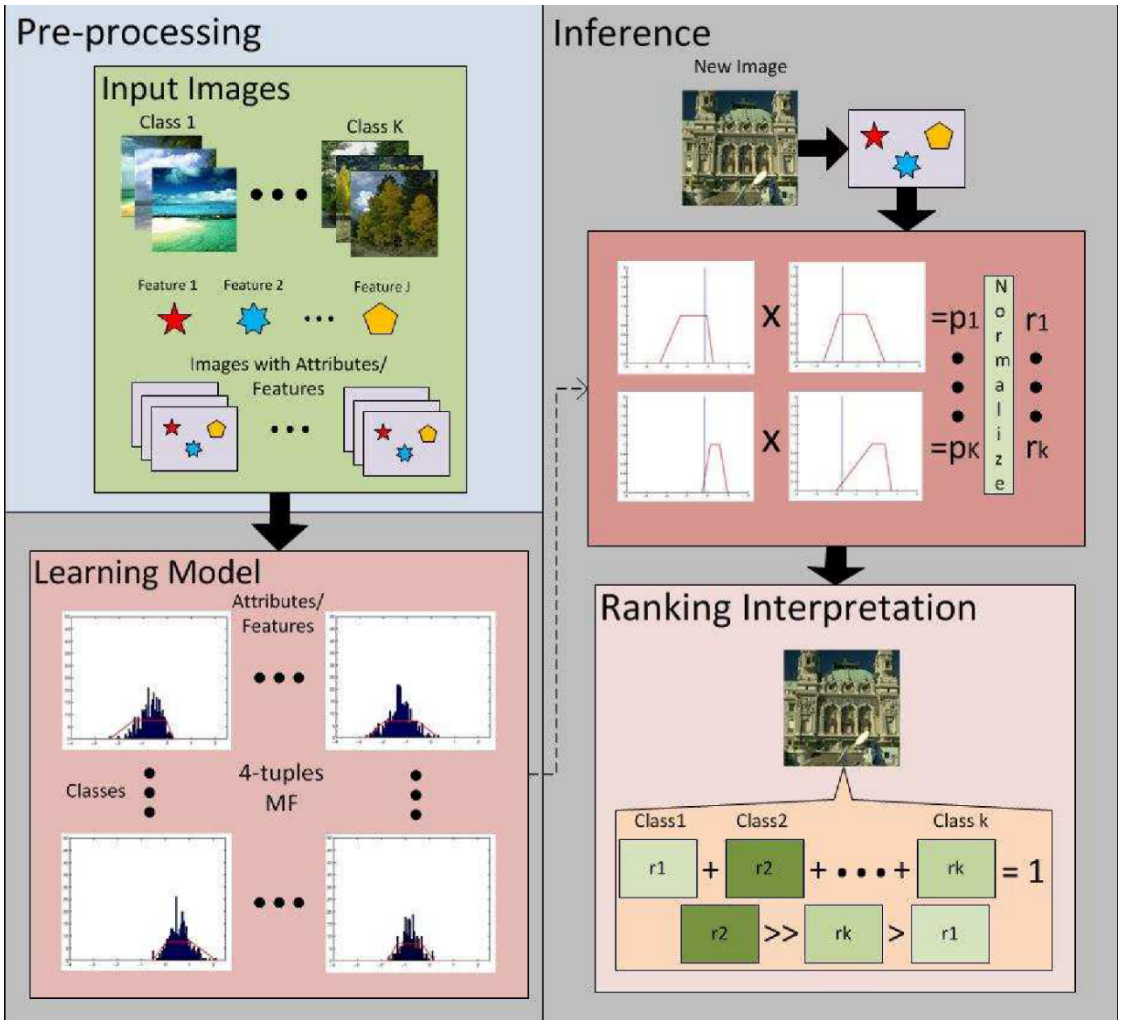



Figure 12. Overall steps for FQRC (image adopted from ***[Lim et al., 2014]***)

The pre-processing step also known as feature extraction step that has been explained in the previous session. The FQRC can support most of the feature descriptors such as LBP, SIFT, and SURF with variance in performances depend on the nature of the application. This toolbox mainly focus on the automated membership generation and the inferencing.

## (a) Fuzzy membership generation (Data driven)

First, the toolbox will generate the membership function for every feature dimension (number of feature dimension depends on the feature descriptor). Theoretically, in the learning model, it learns the image data with parametric approximation of the membership function where the membership distribution of a normal convex fuzzy number is approximated by the 4-tuple, $mf = [a\ b\ \alpha\ \beta]$ ***[Liu & Coghill, 2009]*** with the condition $a < b$ and $ab > 0$. There will be $J \times K$ matrix containing 4-tuple for each feature number, $j$ and class, $k$ denoted by $fqmf$. Those 4-tuples are represented in the form as $mf_{jk}$.

$$fqmf = \begin{bmatrix} mf_{11} & \cdots & mf_{1K} \\ \vdots & \ddots & \vdots \\ mf_{J1} & \cdots & mf_{JK} \end{bmatrix}$$

Nonetheless, histogram analysis is chosen to learn the 4-tuple fuzzy number as illustrated in Fig. 13.

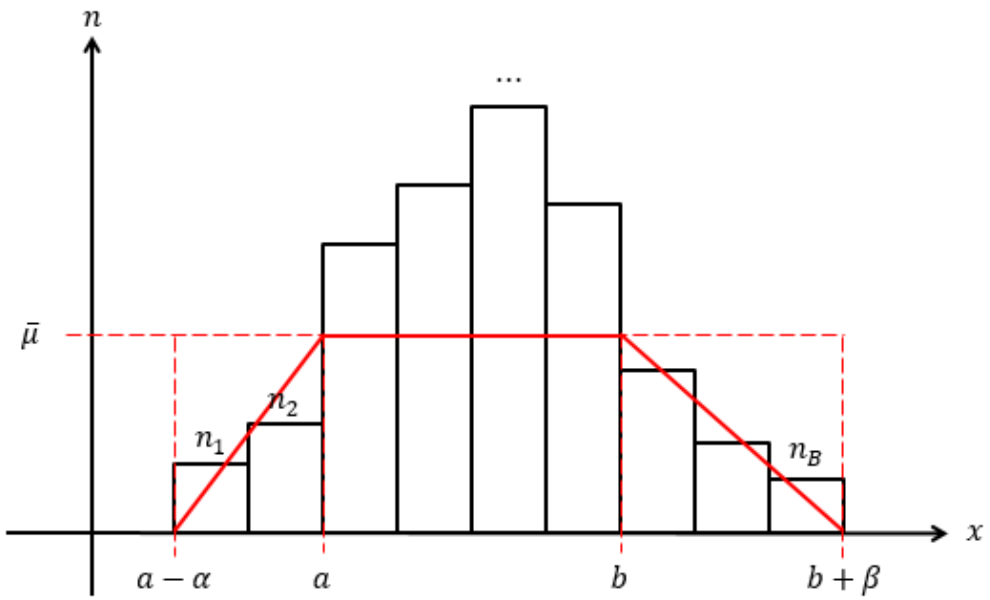



Figure 13. membership generation from histogram

Figure 13 illustrates the parametric representation of a histogram, $x$ is the feature value, $n$ denotes the occurrence of training data from its respective bin $n_1$, $n_2$, … $n_B$. $a$ and $b$ represent the lower and upper bound of $\bar{\mu}$, while $a-\alpha$, and $b+\beta$ represent the minimum and maximum of $x$ value. The dominant region is the area of $[a, b]$.

Where,

$$\bar{\mu} = \frac{\sum_{i=1}^{B} n_i}{b}$$ , with $n_i > 0$

In the toolbox, the function that build the membership function is:

```
build4tuplesMF(feaMat , binNum)
```

Method Implementation:

```
def build4tuplesMF(feaMat , binNum):

    # Parameters initialization
    B = binNum #num of bin
    feaMat = feaMat #feature matrix
    J = feaMat.shape[1] #num of features (also indicate that num of 4-tuples
membership function will be builded at the end)
    mf = np.zeros((J,4))
    mu = [];

    for j in range(0,J): #start building MF
        # calculate the bin width, v
        v = (float(np.amax(feaMat[:,j])) - float(np.amin(feaMat[:,j]))) / float(B)

        # count the ocurrence of the data in the bin and represent in histogram
        h = np.histogram(feaMat[:,j],B);
        N = h[0]
        xout = h[1]
```

```
        # calculate how many bins which have distributed data > 0 (denoted as b)
        b = 0
        for n in range(0,len(N)):
            if (N[n] > 0):
                b = b + 1;

        # Calculate mean value for the histogram
        histMean = float(sum(N)) / b;


        """
        % Find 4-tuple trapezoid position from histogram
        %      _________
        %    /|        |\
        %   / |        | \
        %  /  |        |  \
        % c   a        b   d
        %
        % a-c : alpha
        % d-b : beta
        % 4-tuple = [a,b,alpha,beta]
        """

        # Scan from left to right to obtain a value
        for n in range(0,len(N)):
           if(N[n] >= histMean):
               a = xout[n] #include the offset to get the lower boundary of that bar
               break

        # Scan from right to left to obtain b value
        for n in range(len(N)-1,-1,-1):
            if(N[n] >= histMean):
                b = xout[n+1] #include the offset to get the upper boundary of that
bar
                break

        # obtain a value
        c = xout[0] - v;

        # obtain b value
        d = xout[len(xout)-1] + v;

        # compute alpha
        alpha = a - c;

        # compute beta
        beta = d - b;

        # Output
        mf[j,:] = [a,b,alpha,beta]
        mu.append([histMean, N, xout])

    return mf,mu
```

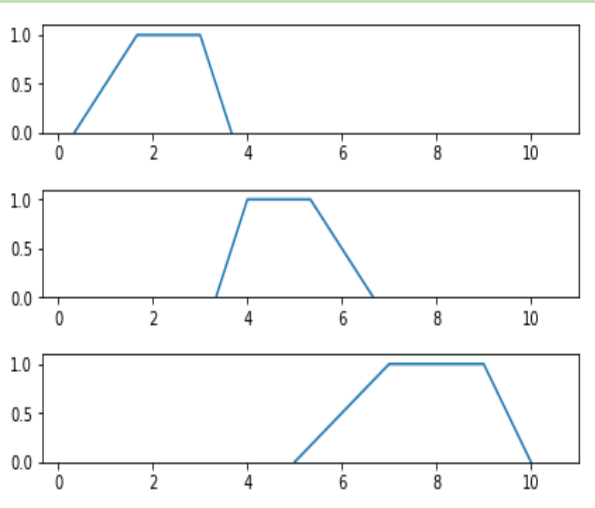
Besides that, the toolbox also provides the visualization facility to view your generated membership functions or the histograms.

```
mfVisualize(fqmf)

histVisualize(fqmf, fqmf_mu)
```

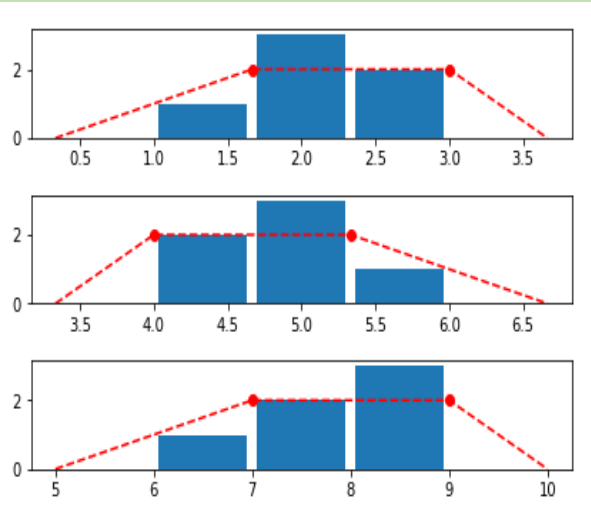
As an example, if I want to generate the membership functions for the feature vectors with **three** dimenions from **six** inputs :

```
Sample code:
a = np.array([(1,4,6),(2,5,7),(2,5,8),(3,4,8),(3,5,7),(2,6,9)])
mf,mu = build4tuplesMF(a,3)
mfVisualize(mf)
histVisualize(mf,mu)
```

```
Output:

mf =
array([[ 1.66666667,  3.        ,  1.33333333,  0.66666667],
       [ 4.        ,  5.33333333,  0.66666667,  1.33333333],
       [ 7.        ,  9.        ,  2.        ,  1.        ]])

Mu =
[[2.0,
  array([1, 3, 2], dtype=int64),
  array([ 1.        ,  1.66666667,  2.33333333,  3.        ])],
 [2.0,
  array([2, 3, 1], dtype=int64),
  array([ 4.        ,  4.66666667,  5.33333333,  6.        ])],
 [2.0, array([1, 2, 3], dtype=int64), array([ 6.,  7.,  8.,  9.])]]


Computer Visualization:
```



(Membership functions)



(Histograms)

## (b) Inference

The goal here is to relax the mutually-exclusive assumption on the image data and classify an unknown scene class into their possibility classes instead of just one. This is unlike the conventional fuzzy inference engine that the defuzzification step eventually derives a crisp decision. Given a testing image and its respective feature values $x$, the membership value μ of feature $j$ belong to class $k$ can be approximated by

$$\mu_{jk}(x_j) = \begin{cases} 0, & x_j < a - \alpha \\ \alpha^{-1}(x_j - a + \alpha), & a - \alpha \leqslant x_j < a \\ 1, & a \leqslant x_j \leqslant b \\ \beta^{-1}(b + \beta - x_j), & b < x_j \leqslant b + \beta \\ 0, & x_j > b + \beta \end{cases}$$

This is then used to calculate the product, $P_k$ of membership values of all the attributes for $k$ class, next in the reference, to generate the ranking result, $P_k$ is normalized against the sum of $P$ of all classes and denoted as $r_k$ .

$$P_k = \prod_{j=1}^{J} \mu_{jk}(x_j)$$

$$r_k = \frac{P_k}{\sum P} = \frac{\prod_{j=1}^{J} \mu_{jk}(x_j)}{Z}$$

In the toolbox, the inference result can be obtained by first determine the membership value:

```
membershipVal(feature_value, mf)
```

Method Implementation:

```
def membershipVal(fvalue, mf):
    # mf -> 4-tuples number retrieve from FQRC (mf = [a b alpha beta])
    a = mf[0]
    b = mf[1]
    alpha = mf[2]
    beta = mf[3]

    if (fvalue >= a and fvalue <= b):      # f_value within [a,b]
        degreeMF = 1
    elif (fvalue >= a-alpha and fvalue < a):    # x within [a-alpha,a]
        degreeMF = (fvalue - a + alpha) / alpha
    elif (fvalue > b and fvalue <= b+beta):   # x within [b,b+beta]
        degreeMF = (b + beta - fvalue) / beta
    else:
        degreeMF = 0

    return degreeMF
```

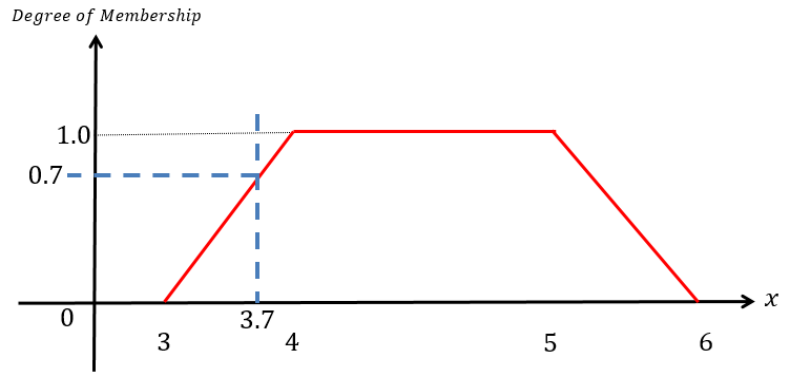
As an example:

Sample code:

```
mf = np.array([4, 5, 1, 1])
membership_degree = membershipVal(3.7, mf)
```

Output:

```
membership_degree = array([ 0.7 ])
```




then followed by the inference method:

```
inference( feaVec, fqmf)
```

Method Implementation:

```
def inference( feaVec, fqmf):
    J = len(fqmf[0]) #num of feature
    K = len(fqmf) #num of class

    # Obtain degree of membership for each feature value
    feaDegreeMF = np.zeros((K,J))

    for k in range(0,K):
        for j in range(0,J):
            degreeMF = membershipVal(feaVec[j], fqmf[k][j]);
            feaDegreeMF[k][j] = degreeMF;

    temp = copy.copy(feaDegreeMF)
    temp[feaDegreeMF > 0] = 1
    hitCount = np.sum(temp,axis=1)
    sumOfdegreeMF = np.sum(feaDegreeMF,axis=1)
    ratio = np.divide(sumOfdegreeMF, np.amax(hitCount))
    sumRatio = sum(ratio)
    normOfdegreeMF = np.divide(ratio,sumRatio) #Normalization
    output = normOfdegreeMF

    if(np.isnan(sum(output))==True or np.isinf(sum(output))==True):
        output = np.zeros((1,K))
    return output, feaDegreeMF
```

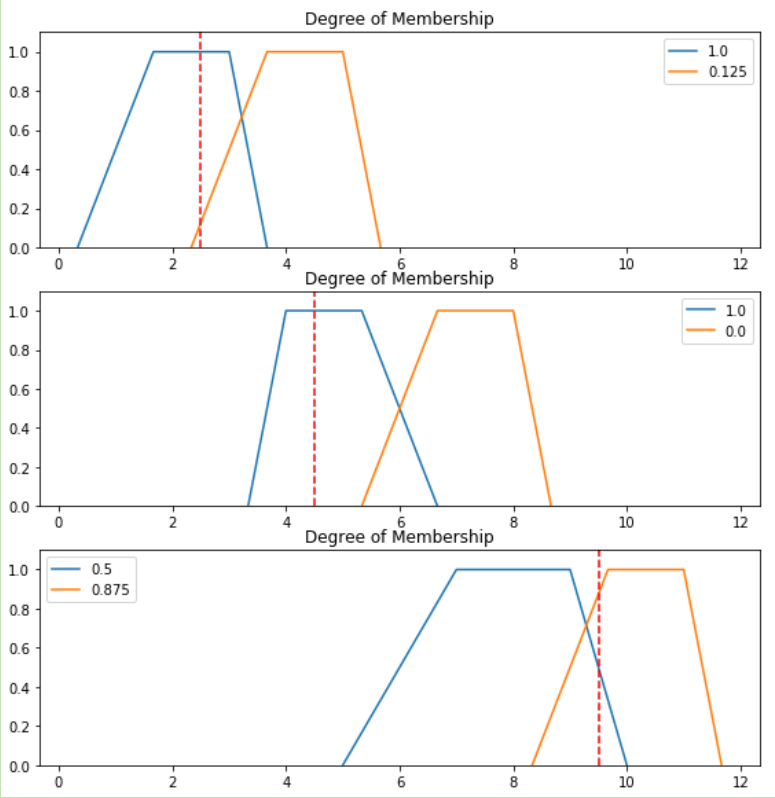
Similarly, the visualization to inspect the cross over degree of membership for each feature corresponds to all classes is provided.

```
infVisualize(fqmf)
```

As an example:

```
Sample code:
a = np.array([(1,4,6),(2,5,7),(2,5,8),(3,4,8),(3,5,7),(2,6,9)]) # Class 1
mf_a,mu_a = build4tuplesMF(a,3)

b = np.array([(4,6,9),(4,7,10),(3,7,11),(5,7,10),(4,8,10),(5,8,11)]) # Class 2
mf_b,mu_b = build4tuplesMF(b,3)

mf_all = [] # To append membership functions for all classes
mf_all.append(mf_a)
mf_all.append(mf_b)

feaVec = np.array([2.5,4.5,9.5]) # New input with feature values
output,feaDegreeMF = inference(feaVec, mf_all)
infVisualize(feaVec, mf_all, feaDegreeMF)
```

```
Output:

output = array([ 0.71428571,  0.28571429])

feaDegreeMF =
array([[ 1.   ,  1.   ,  0.5  ],
       [ 0.125,  0.   ,  0.875]])
```



## (c) Classification using FQRC

To ease the user, the classification using FQRC is simplified with the train function "`CL_FQRC_Train`" which yield the membership functions (fqmf) for all features and classes, and "`CL_FQRC_Predict`" to predict a new input with using the fqmf generated from the training step. User can choose to visualize the result by inputing "`visualize=True`".

```
CL_FQRC_Train(X_train, y_train, binNum, visualize)

CL_FQRC_Predict(X_test, fqmf, visualize)
```

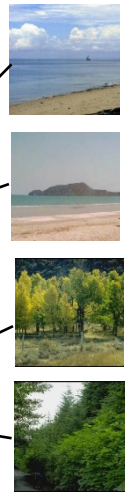
In the "`CL_FQRC_Train`" function, user is require to provide the feature vectors obtained from the training data (X_train) and the ground truth for each data (y_train) as an array.

X train =

| $Feature_{j=1}$ | $Feature_{j=2}$ | $Feature_{j=3}$ |
|---|---|---|
| 1 | 4 | 6 |
| 2 | 5 | 7 |
| | . . . | |
| 4 | 8 | 10 |
| 5 | 8 | 11 |

y_train =

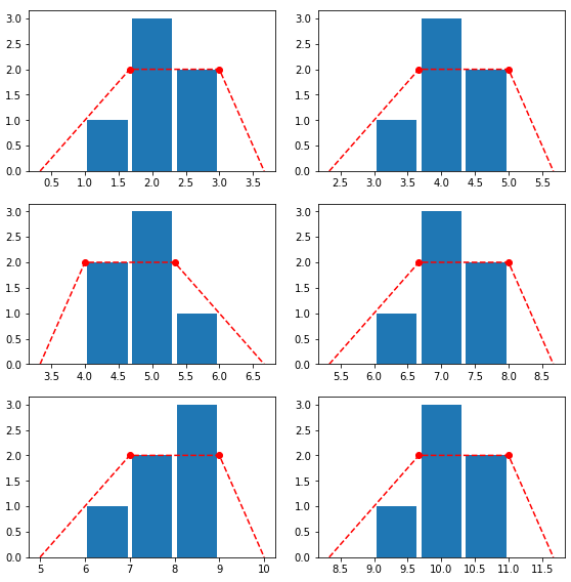
| GroundTruth |
|---|
| 0 |
| 0 |
| . . . |
| 1 |
| 1 |

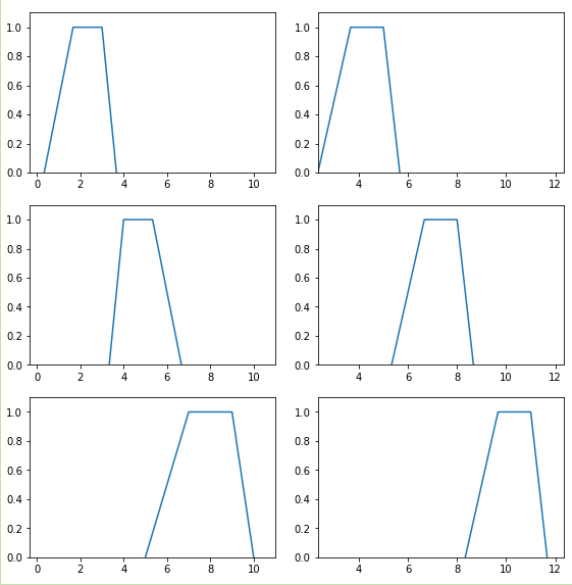


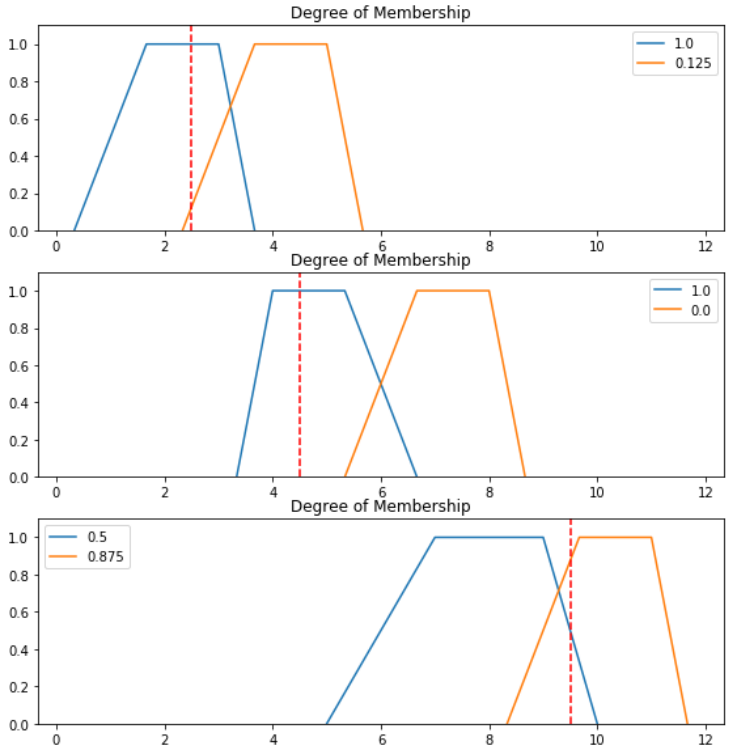
Sample code:

```
X_train =
np.array([(1,4,6),(2,5,7),(2,5,8),(3,4,8),(3,5,7),(2,6,9),(4,6,9),(4,7,10),(3,7,11),(5
,7,10),(4,8,10),(5,8,11)])
x_groundTruth = np.array([0,0,0,0,0,0,1,1,1,1,1,1])
fqmf = CL_FQRC_Train(X_train, x_groundTruth, 3, True)

feaVec = np.array([2.5,4.5,9.5]) %--> new testing input
result = CL_FQRC_Predict(feaVec, fqmf, True)
print 'output:' + str(output)
```

Output:








```
result: [0.71428571  0.28571429]
```

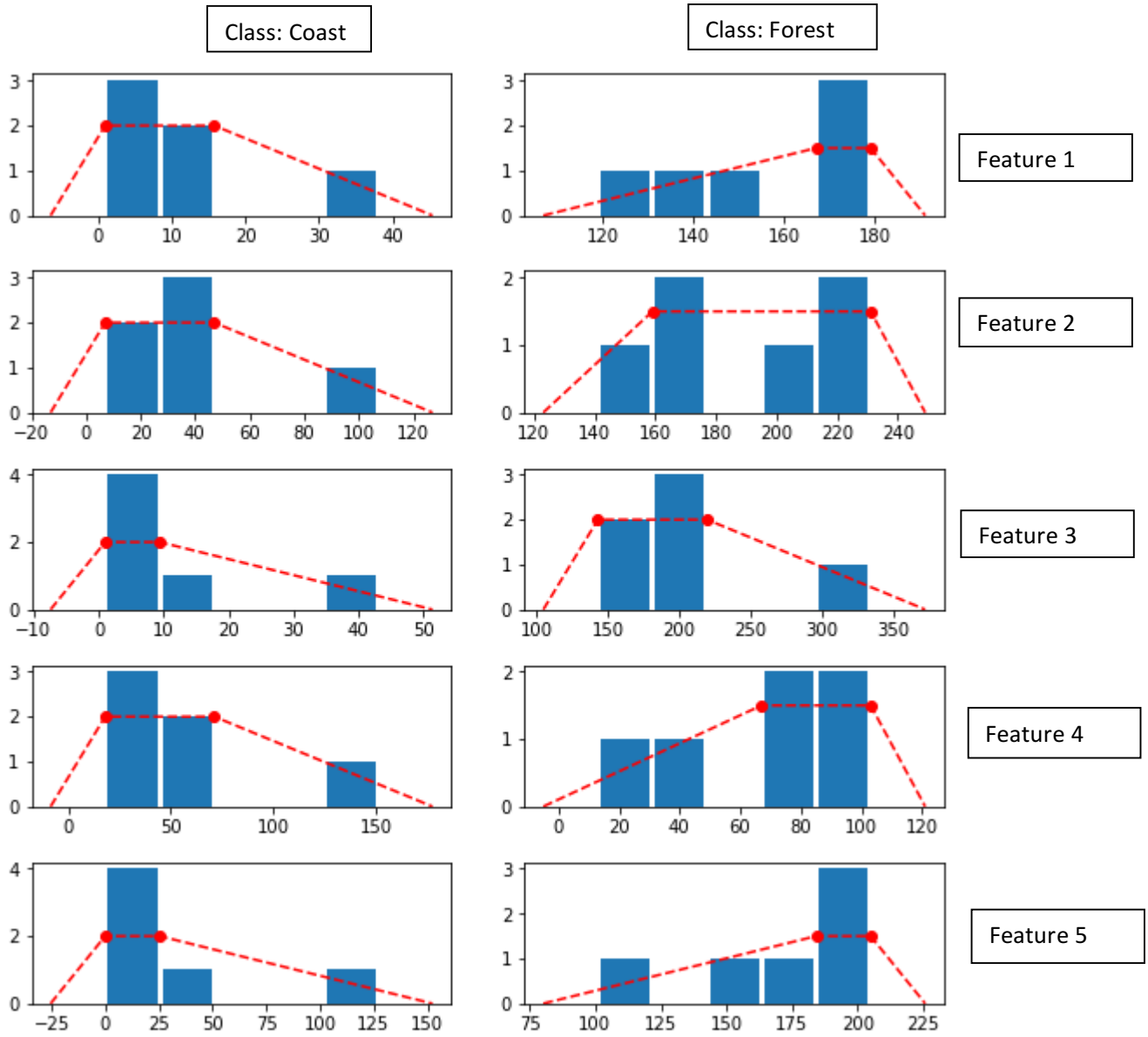
One can notice that the result generated he is not a crisp outcome but the membership (confident) value of the new input belongs to the respective class.

## (d) Image Classification with Fuzzy Approach

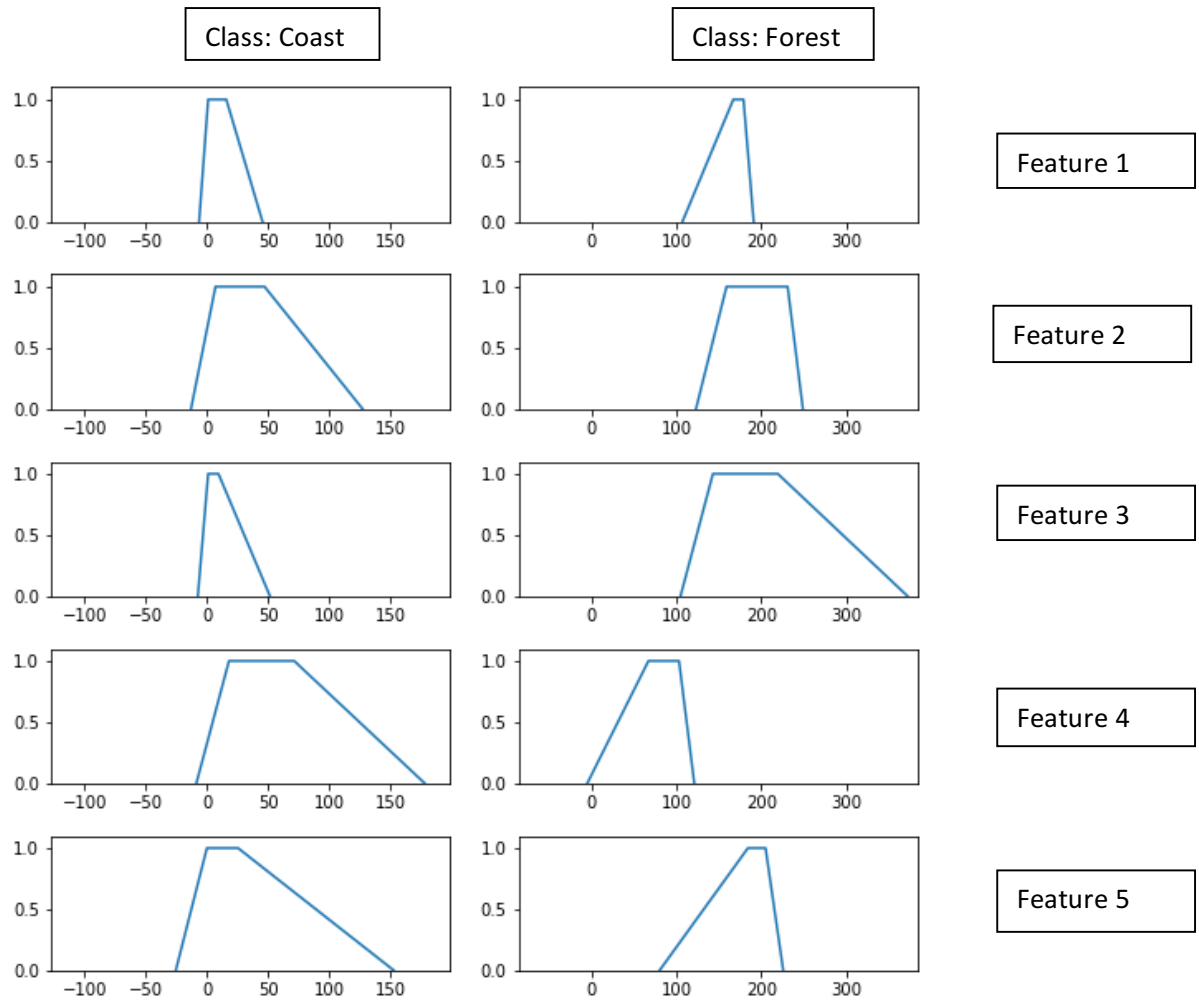
User can use the following code to perform image classification with the FQRC.

```
Sample code:
Output_fuzzy = fcvt.Image_Classification('Training', 'Testing', 'SIFT', 'Fuzzy')
```

Using the same case study in scene understanding, first the membership function for each feature and class is generated using histogram. The output is shown as Figure X, and Figure X. This example is using SIFT feature with BoF of five clusters (five different codewords in the codebook).

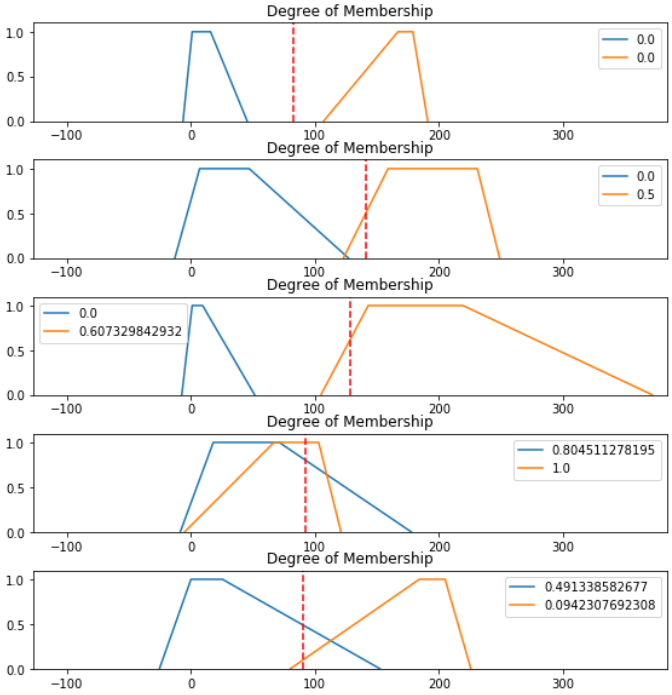



Based on the number of class in the train folder, the system automatically generate the corresponding membership function that represent each feature dimension for each class.

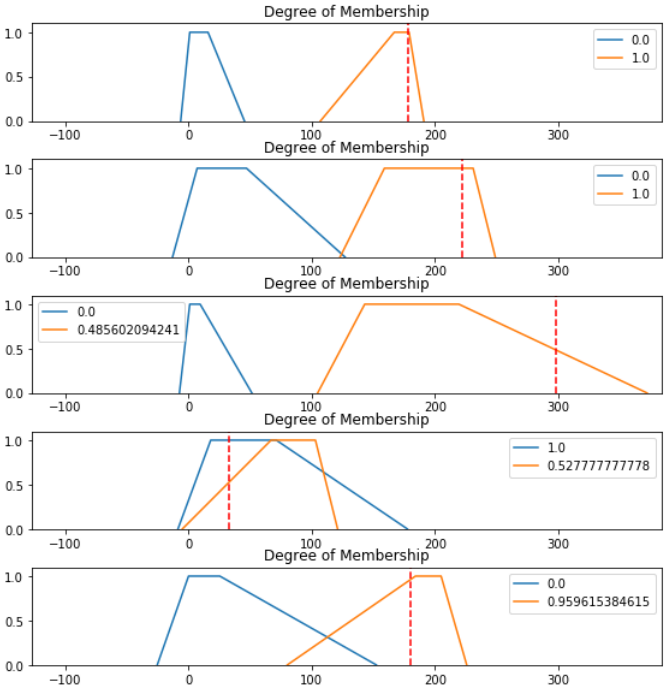


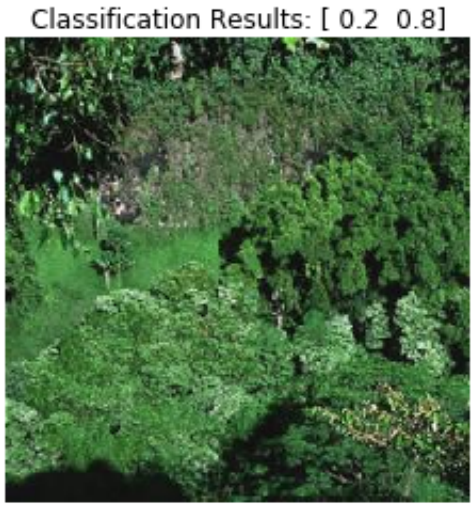

Based on the generated membership functions, inference can be done on the testing images. The sample output are as follows:

`Output:`








One of the strengths of the FQRC is it provides the feasibility to perform single-label classification task like other crisp classifiers as well as ranking as shown in the results.

* More results can be observed from the python program.

# References:

Bay, H., Ess, A., Tuytelaars, T., & Van Gool, L. (2008). Speeded-up robust features (SURF). *Computer vision and image understanding*, *110*(3), 346-359.

Canny, J. (1986). A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, (6), 679-698.

Fei-Fei, L., & Perona, P. (2005, June). A bayesian hierarchical model for learning natural scene categories. In *IEEE International Conference on Computer Vision and Pattern Recognition,* Vol. 2, pp. 524-531).

Harris, C., & Stephens, M. (1988, August). A combined corner and edge detector. In *Alvey vision conference* (Vol. 15, No. 50, pp. 10-5244).

He, D. C., & Wang, L. (1990). Texture unit, texture spectrum, and texture analysis. *IEEE transactions on Geoscience and Remote Sensing*, *28*(4), 509-512.

Lim, C. H., Risnumawan, A., & Chan, C. S. (2014). A scene image is nonmutually exclusive—a fuzzy qualitative scene understanding. *IEEE Transactions on Fuzzy Systems*, *22*(6), 1541-1556.

Liu, H., & Coghill, G. M. (2005, October). Fuzzy qualitative trigonometry. In *2005 IEEE International Conference on Systems, Man and Cybernetics,* (Vol. 2, pp. 1291-1296).

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, *60*(2), 91-110.

Oliva, A., & Torralba, A. (2001). Modeling the shape of the scene: A holistic representation of the spatial envelope. *International journal of computer vision*, *42*(3), 145-175.

Otsu, N. (1979). A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics*, *9*(1), 62-66.

Salton, G., & McGill, M. J. (1983). Introduction to modern information Philadelphia, PA. American Association for Artificial Intelligence retrieval.

Zadeh, L. A. (1965). Fuzzy sets. *Information and control*, *8*(3), 338-353.