Chern Hong Lim

chlim@acd.tarc.edu.my / lch8743@hotmail.com

Abstract

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

**Tutorial:   
Fuzzy Computer Vision Toolbox**

FUZZ 2017

Table of Contents

[**Pre-requisite** 2](#_Toc486887102)

[**Introduction** 3](#_Toc486887103)

[**Section A: Image Acquisition** 5](#_Toc486887104)

[**(1) Read Source** 5](#_Toc486887105)

[**Section B: Image Pre-Processing** 6](#_Toc486887106)

[**(1): Image resize** 6](#_Toc486887107)

[**(2): Image conversion** 6](#_Toc486887108)

[**(3): Image Morphological Operations** 7](#_Toc486887109)

[**(4): Image Filtering** 8](#_Toc486887110)

[**Section C: Feature Extraction** 10](#_Toc486887111)

[**(1): Visual Image Features** 10](#_Toc486887112)

[(a): Color Detector 10](#_Toc486887113)

[(b): Edge Detector 11](#_Toc486887114)

[(c): Corner detector 11](#_Toc486887115)

[**(2): Holistic Image Features** 12](#_Toc486887116)

[(a): LBP detector 12](#_Toc486887117)

[**(3): Local Image Features** 12](#_Toc486887118)

[(a): Keypoint detector 12](#_Toc486887119)

[**(4): Feature Representation using Bag of Feature (For keypoints detector)** 13](#_Toc486887120)

[**Section D: Image Classification** 15](#_Toc486887121)

[**(1) Crisp Classification** 16](#_Toc486887122)

[(a) SVM Classification 16](#_Toc486887123)

[(b) Image Classification with Crisp 16](#_Toc486887124)

[**(2) Fuzzy Classification (FQRC)** 16](#_Toc486887125)

[(a) Fuzzy membership generation (Data driven) 17](#_Toc486887126)

[(b) Inference 21](#_Toc486887127)

[(c) Classification using FQRC 24](#_Toc486887128)

[(d) Image Classification with Fuzzy 25](#_Toc486887129)

[(c) Limitation 28](#_Toc486887130)

[**Extra(1): Fuzzy Deep Learning** 29](#_Toc486887131)

[**Extra(2): Read video** 30](#_Toc486887132)

[**References:** 32](#_Toc486887133)

# **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)
   1. Download from: <https://www.continuum.io/downloads>
2. Install opencv library (version 2.4.x.x)
   1. Download opencv library from: <http://opencv.org/releases.html>
   2. Double-click to extract the opencv.
   3. Go to “opencv/build/python/2.7/x64 folder.”
   4. Copy cv2.pyd to your python directory in the “lib/site-packages”.
3. Install scikit-image package.
   1. Open anaconda prompt
   2. Type “pip install scikit-image”
   3. Web reference: <http://scikit-image.org/>
4. Install scikit-learn package.
   1. Open anaconda prompt
   2. Type “pip install scikit-learn”
   3. 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) and make 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.

Image preprocessing

Feature Extraction

Classification / Interpretation

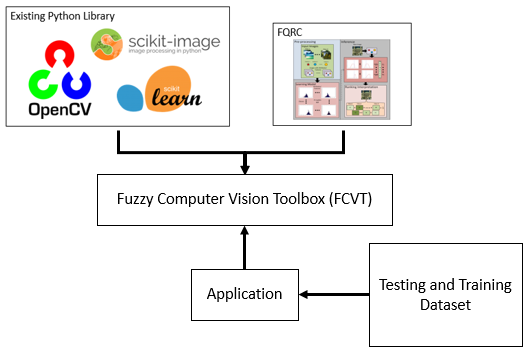
Image/Video Acquisition

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 ***[Hong 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:

1. Image / Video Acquisition
   1. Read Source
2. Image Preprocessing
3. Resize image
4. Image conversion
5. Morphological operation
6. Image filtering
7. Feature Extraction
   1. Color detection
   2. Edge detection
   3. Corner detection
   4. Keypoint Detection
   5. LBP
   6. Feature Representation
8. Image Classification
9. Crisp Classification
10. Fuzzy Classfication

Please note that the fundamental computer vision process 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 self-developed FQRC toolbox. For simplicity, the necessary functions are packaged in a main python library namely FCVT.



***Sample images and the training and testing dataset used in this tutorial are provided.***

***Installation guide is provided***

***Examples used in this tutorial are provided:***

***Main.py***

***Python file is provided:***

***FCVT.py***

***Python file is provided:***

***FQRC.py***

Figure 1: Overview of FCVT

User can import FCVT to access all the functions that 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.

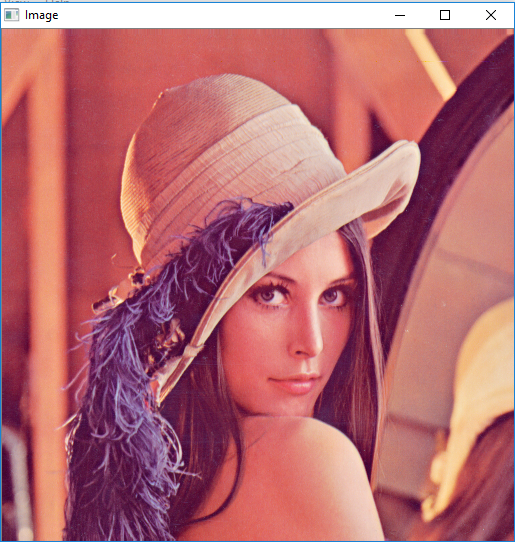
IA\_readSource( sourceDir, display )

Sample Code:

sourceDir = 'Lenna.png'

image = fcvt.IA\_readSource(sourceDir, display=True)

Output:



# **Section B: Image Pre-Processing**

## **(1): Image resize**

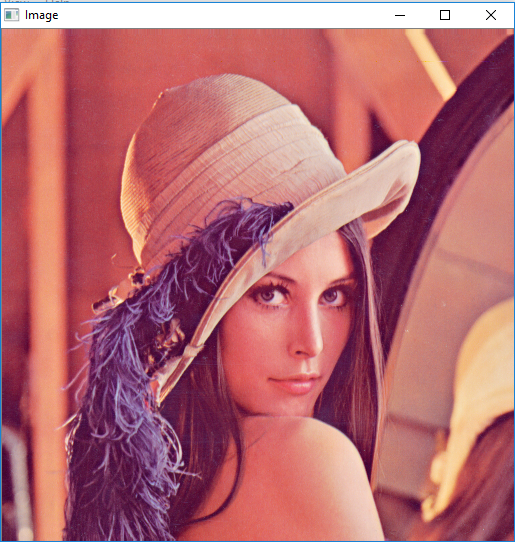
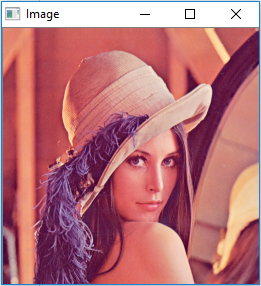
An image can be resized with the “IP\_resize” function where “sx” and “sy” indicate the scale that the image to be resized (Example: 0.5 means to downsize the image into ½ of the original size).

IP\_resize(image, sx, sy, display)

Sample Code:

imageResize = fcvt.IP\_resize(image, 0.5, 0.5, True)

Output:

## 

## **(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, N., 1979]***.

IP\_convertGray(image, display)

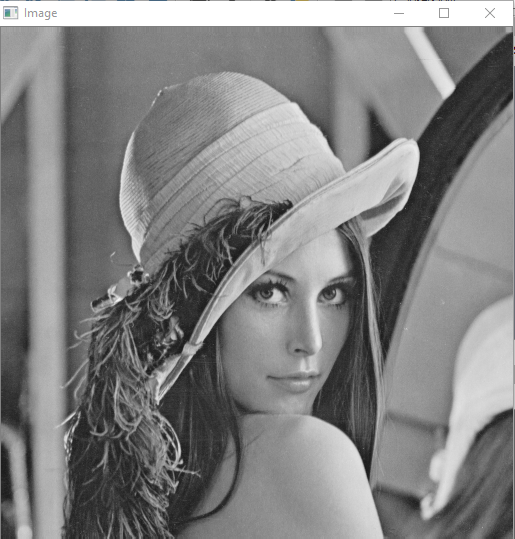
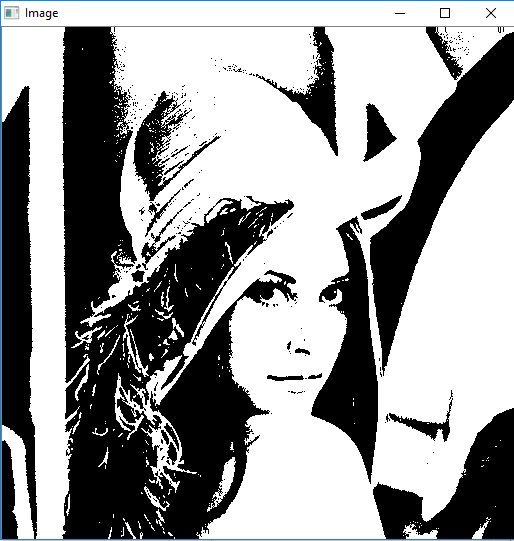
IP\_convertBinary(image, display)

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 1: Dilation with structuring element as (b)

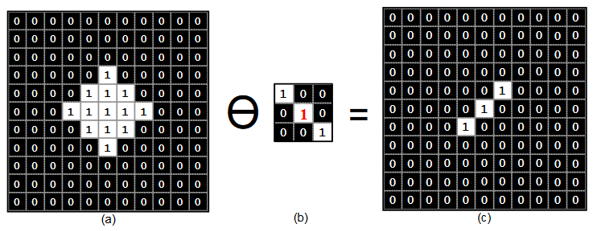


Figure 2: Erosion with structuring element as (b)

Following lines perform morphological operation with a Digit image.

\*Note: The method argument can be the followings:

1. erosion
2. dilation
3. open (Erode then dilate)
4. 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)

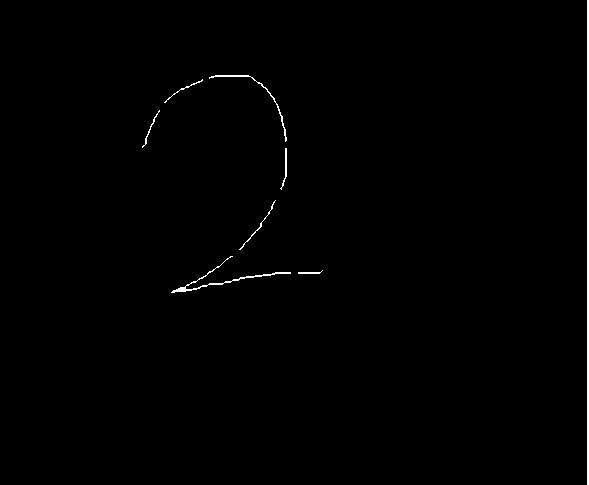
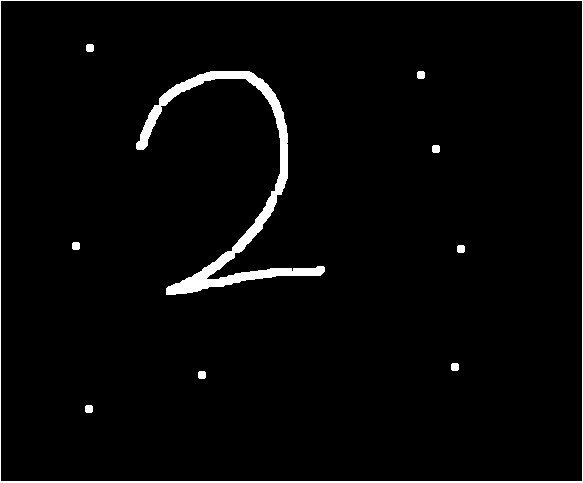
imageMorph = fcvt.IP\_imageMorph(imageDigit, 'dilation', (5,5), True)

imageMorph = fcvt.IP\_imageMorph(imageDigit, 'opening', (3,3), True)

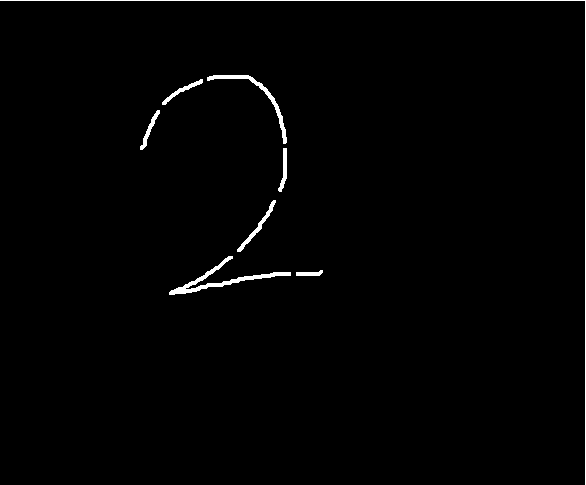
imageMorph = fcvt.IP\_imageMorph(imageDigit, 'closing', (5,5), True)

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 imaged) 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:

1. average
2. gaussian
3. 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:

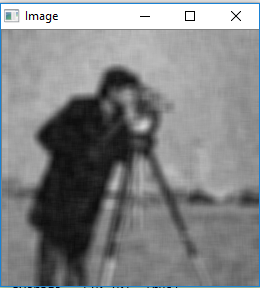
imageCameraman = fcvt.IA\_readSource('cameraman\_noise.jpg', True)

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)

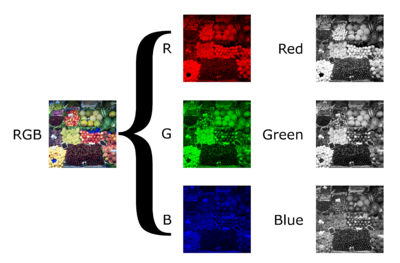
 

# **Section C: Feature Extraction**

## **(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 3: 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 elements from the map images (“map.png”) 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 lowebound and upperbound in the example indicate the python list with [R, G, B] value.

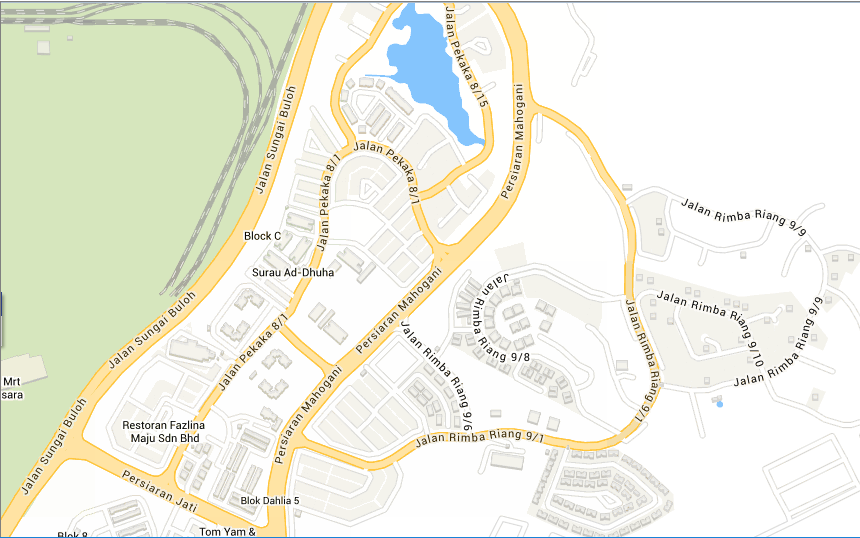
FE\_colorDetection(image, lowerbound, upperbound, display)

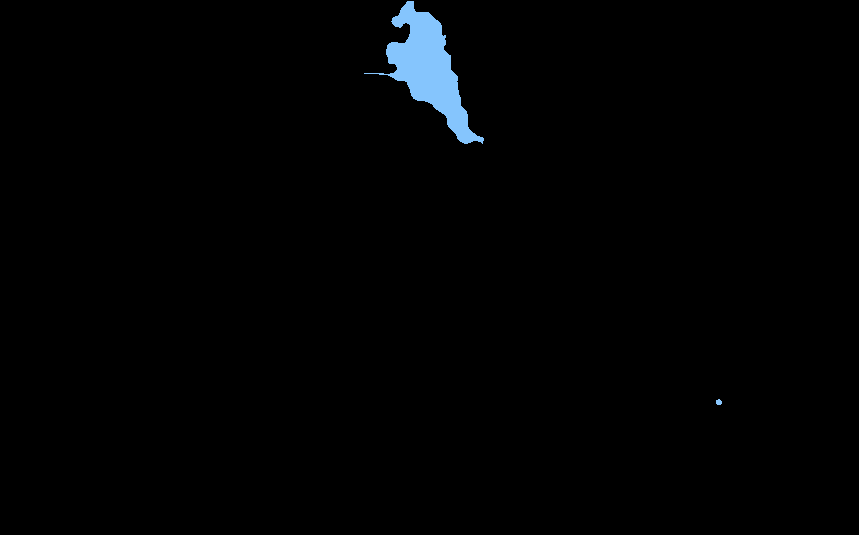
Sample Code:

imageMap = fcvt.IA\_readSource('map.png', True)

imageColor = fcvt.FE\_colorDetection(imageMap, lowerbound, upperbound, True)

Output:



lowerbound = [120,185,240] lowerbound = [245,205,100]

upperbound = [150,205,260] upperbound = [265,230,170]

lowerbound = [204,220,179] lowerbound = [235,214,228]

upperbound = [224,240,199] 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, J., 1986]*** to extract the edge of the Chessboard image.

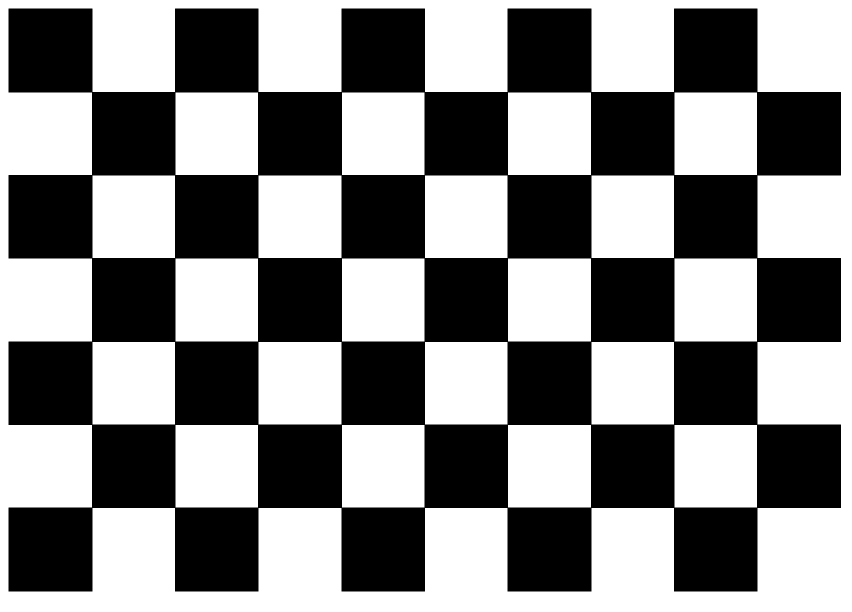
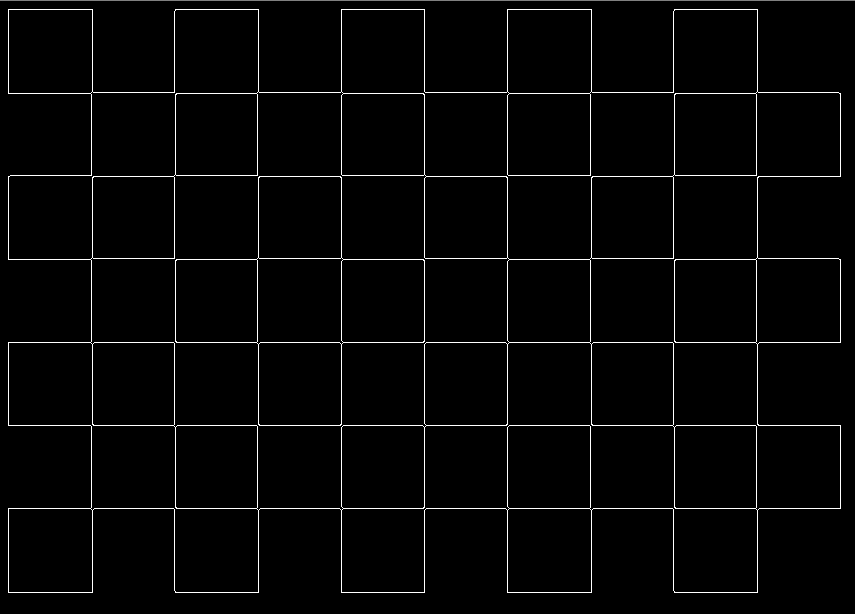
FE\_edgeDetection(image, method, display)

Sample Code:

imageChessboard = fcvt.IA\_readSource('imageChessboard.png', True)

imageEdge = fcvt.FE\_edgeDetection(image, 'canny', 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 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.

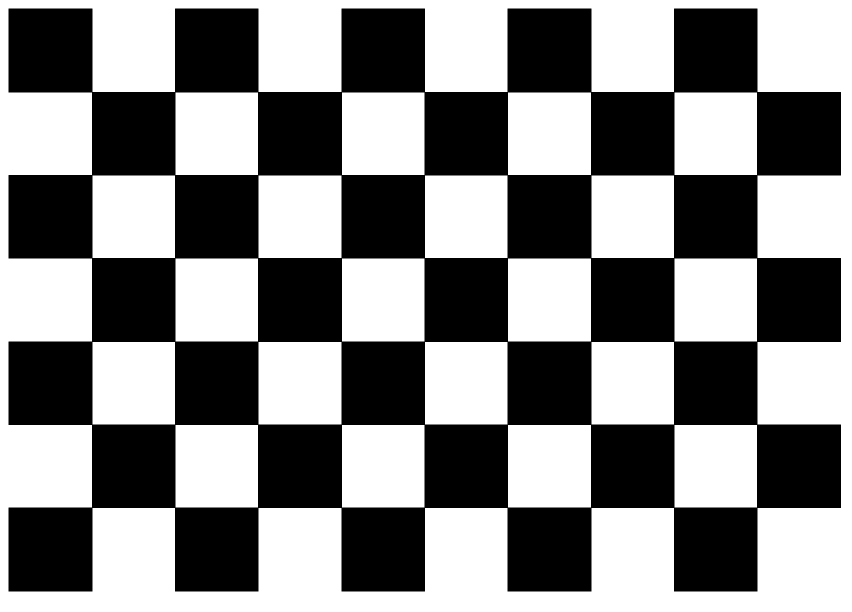
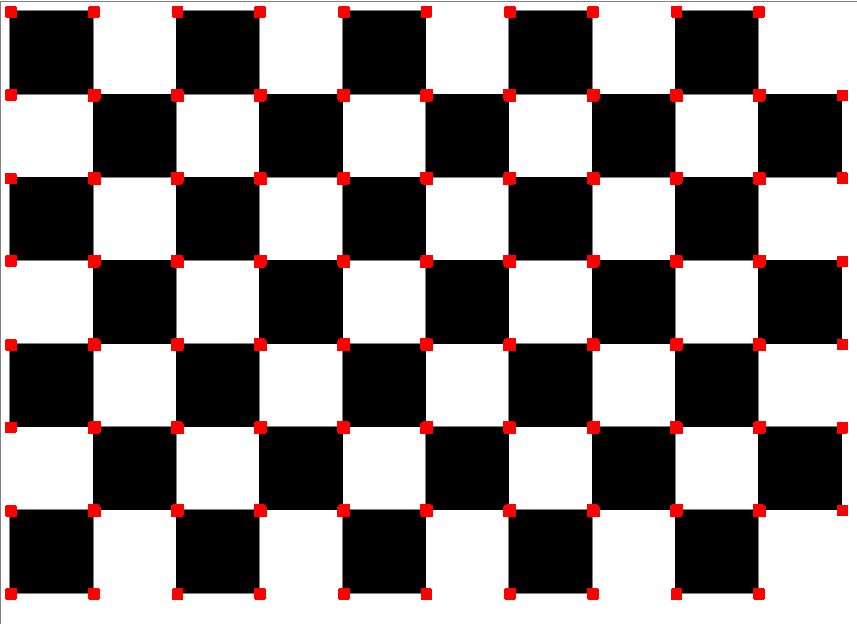
FE\_cornerDetection(image, display)

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): LBP detector

What is LBP?

FE\_LBPDetection(image, method, display)

Sample Code:

imageLBP = fcvt.FE\_LBPDetection(image, 'uniform', True)

Output:

## **(3): Local Image Features**

### (a): Keypoint detector

For any object in an image, interesting points on the object can be extracted to provide a "feature description" of the object. This description, 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. To perform reliable recognition, it is important that the features extracted from the training image be detectable even under changes in image scale, noise and illumination. Such points usually lie on high-contrast regions of the image, such as object edges.

Another important characteristic of these features is that the relative positions between them in the original scene shouldn't change from one image to another. For example, if only the four corners of a door were used as features, they would work regardless of the door's position; but if points in the frame were also used, the recognition would fail if the door is opened or closed. Similarly, features located in articulated or flexible objects would typically not work if any change in their internal geometry happens between two images in the set being processed. SIFT [Lowe, 2004], and SURF [Bay, et. al., 2008]

\*Note: The method argument can be the followings:

1. SIFT
2. SURF

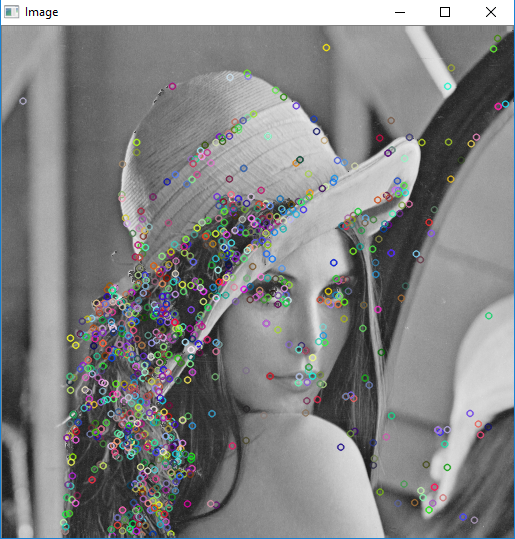
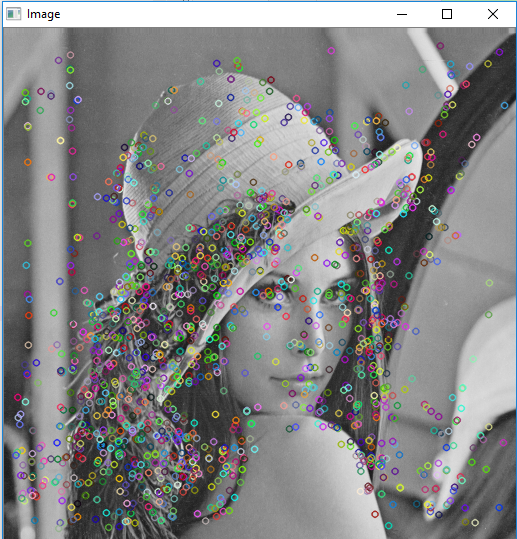
FE\_keypointDetection(image, method, display)

Sample Code:

imageKeyPoint = fcvt.FE\_keypointDetection(image, 'SIFT', True)

imageKeyPoint = fcvt.FE\_keypointDetection(image, 'SURF', True)

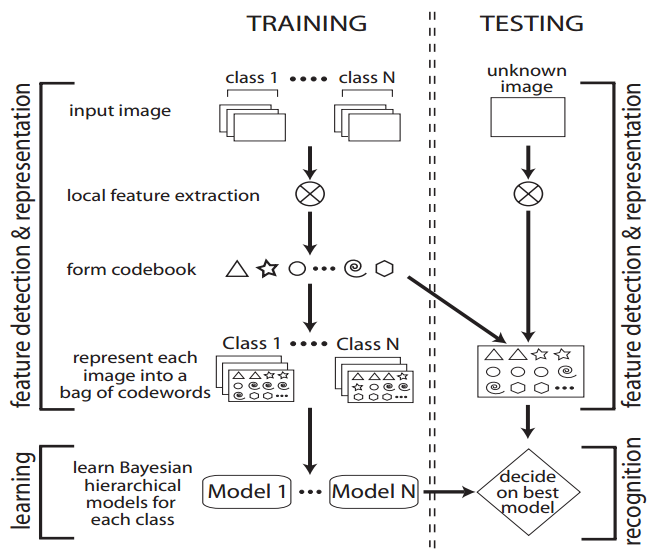
Output:

(SIFT) (SURF)

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

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



Unsupervised learning to form codebook

Supervised Learning

Figure 2. Overall BoF framework. (Image adopted from ***[Li, and Perona, 2005]***)

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 [citation]. The overall process of BoF is shown in Figure X.

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(imageKeyPoint, 'kmeans', 5)

FE\_Quantisation(imageKeyPoint, cluster)

Sample Code:

cluster = fcvt.FE\_Clustering(imageKeyPoint, 'kmeans', 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 problem, which is the task of assigning an input image with label from a fixed set of categories. First, a sufficient amount of training images from different classes are used to train a classifier and the classifier will be used to classify a testing image into respective class.

Training Images:



Class: Coast

Class: Car

Testing Images:

Classification output:

**Coast ? Forest**

The conventional goal of the classification tasks is to assign an unknown scene image to one of the several possible classes. For example, Fig. 1(a) is a Coast class while Fig. 1(c) is a Forest class. Intentionally, most state-of-the-art approaches in image classification are exemplar-based and due with images that 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. 1(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 [citation], this work study the effectiveness of using Fuzzy Qualitative Rank Classifier (FQRC) 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 will be used. In the toolbox, the image classification function is simplified with the following syntax. And

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 method (‘Crisp’ or ‘Fuzzy’).

## **(1) Crisp Classification**

### (a) SVM Classification

Describe briefly about SVM.

### (b) Image Classification with Crisp Approach

For Crisp classification, Support Vector Machine (SVM) is applied in the toolbox.

Sample code:

Output\_crisp = fcvt.Image\_Classification('Training', 'Testing', 'sift', 'Crisp')

Output:

## **(2) Fuzzy Classification (FQRC)**

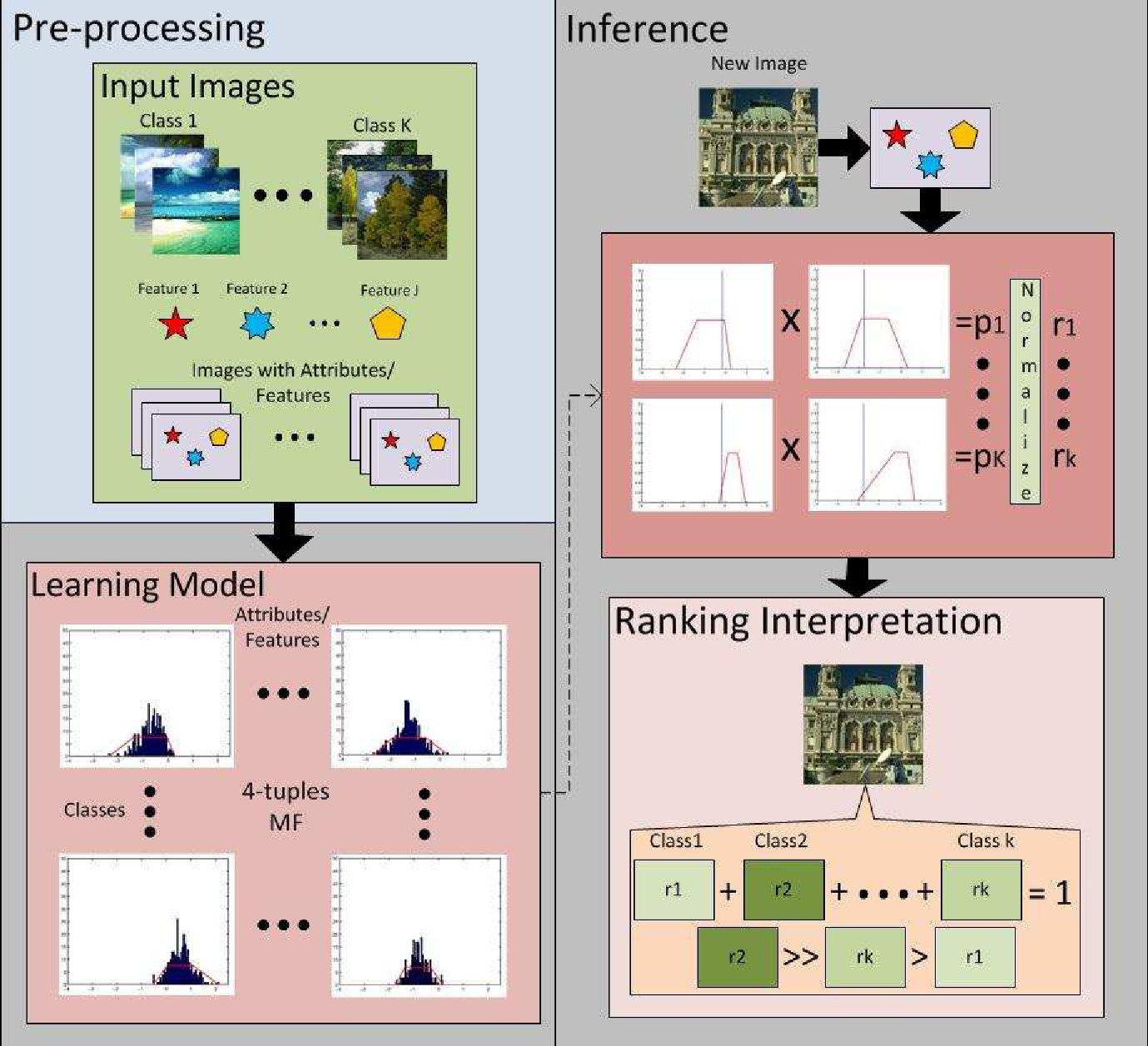
The limitation of the Crisp classification:

1. Unable to deal with uncertainty or ambiguous case

The limitation of the ordinary FIS:

1. Manual / Expert is needed for membership generation
2. Rule is mandatory
3. The inference is based on rules

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 Fig. 2.

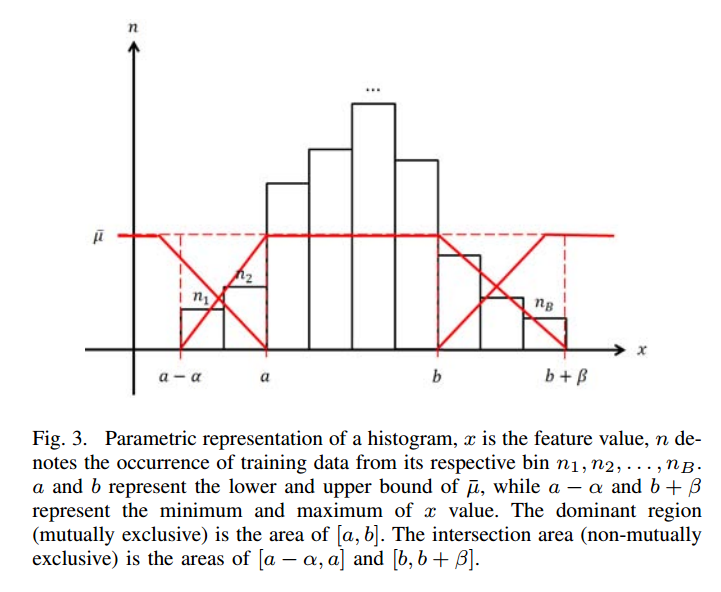


The pre-processing step alson known as feature extraction step has been explained in the previsou session. The FQRC can support most of the feature descriptor such as LBP, HOG, Sift, and Surf with variants performance depends on the nature of the application and the distinguish power of the feature descriptor. With this, the tutorial continue with introducing the automated fuzzy generation mechanism, which is used to learn the model.

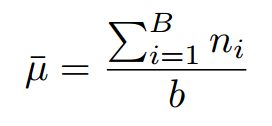
A library named FQRC.py has been implemented to support the above funtions.

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

First, the toolbox will generate the membership function for every feature dimension (determined by the number of cluster in BoF). Theoretically, In the learning model, it learn 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 [a b α β] [citation]. Histogram analysis is chosen to learn the 4-tuple fuzzy number as illustrated in Fig. 3.



Where,



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, 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:

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,

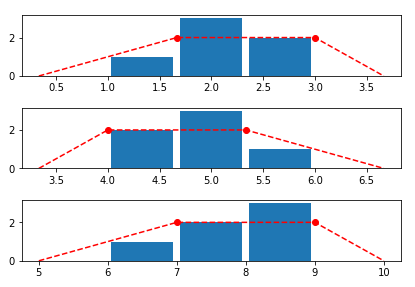
array([2, 3, 1], dtype=int64),

array([ 4. , 4.66666667, 5.33333333, 6. ])],

[2.0, array([1, 2, 3], dtype=int64), array([ 6., 7., 8., 9.])]]

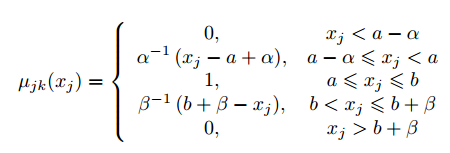
Computer Visualization:



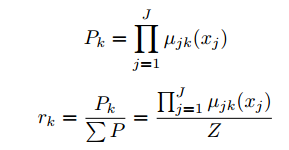


### (b) Inference

The goal here is to relax the mutually-exclusive assumption on the image data and classify an unknown scene class into their possibility scene 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 scene image and its respective feature values x, the membership value µ of feature j belong to class k can be approximated by



This is then used to calculate the product, Pk of membership values of all the attributes for each class, k, and normalize the Pk as rk by using



In the toolbox, the function that perform the inference is first to determine the membership value then followed by inference method:

membershipVal(fvalue, 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:

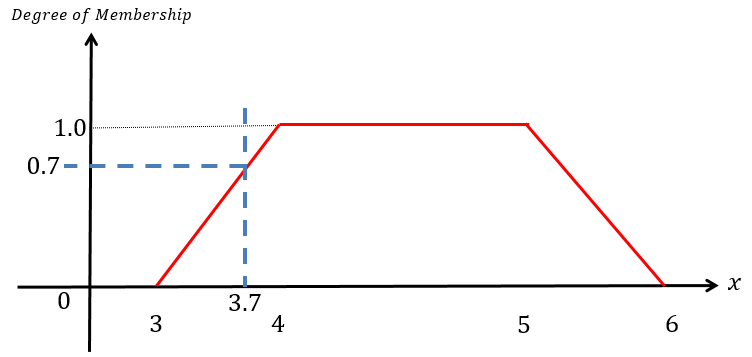
Sample code:

mf = np.array([4, 5, 1, 1])

membership\_degree = **membershipVal**(3.7, mf)

Output:

membership\_degree = array([ 0.7 ])



And for 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:

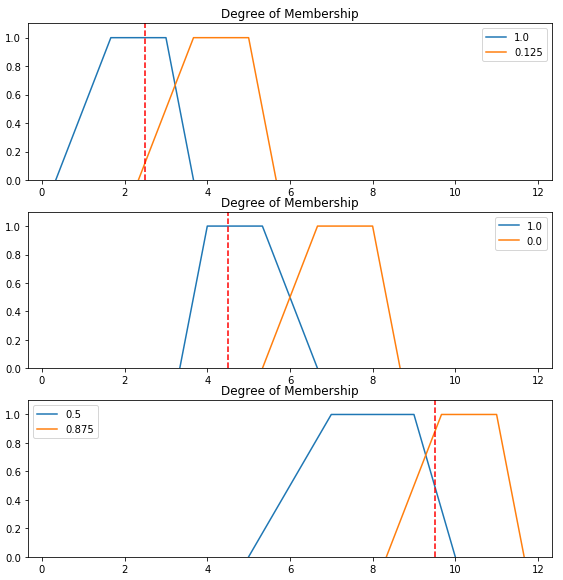
output = array([ 0.71428571, 0.28571429])

feaDegreeMF =

array([[ 1. , 1. , 0.5 ],

[ 0.125, 0. , 0.875]])

Computer Visualization:



### (c) Classification using FQRC

To ease the user, the classification using FQRC is simplified as following:

CL\_FQRC\_Train(X\_train, y\_train, binNum, visualize)

CL\_FQRC\_Predict(X\_test, fqmf, visualize)

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])

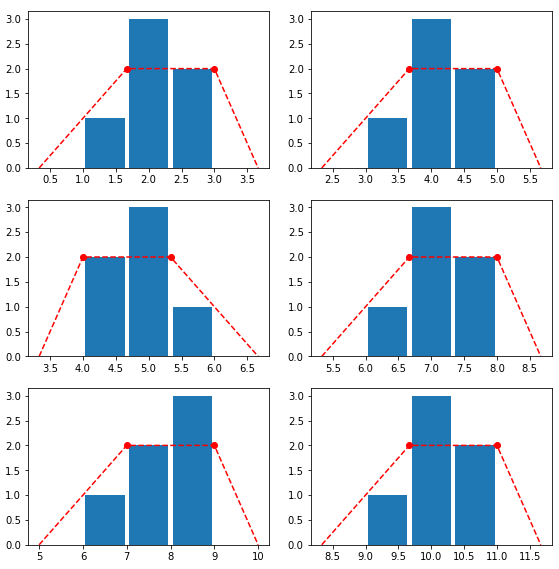
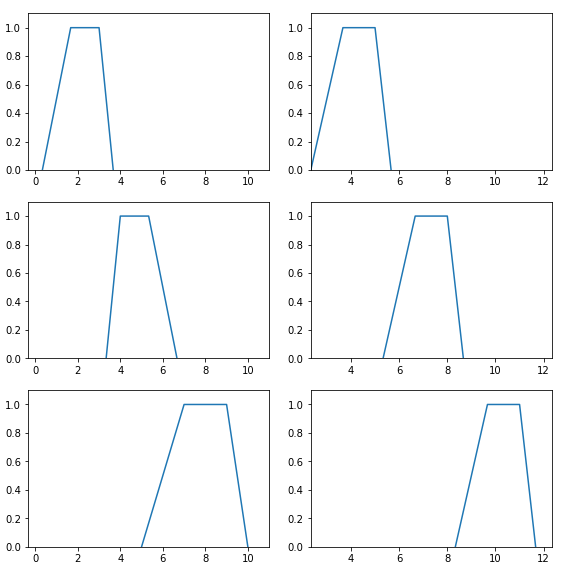
fqmf = **CL\_FQRC\_Train**(X\_train, x\_groundTruth, 3, True)

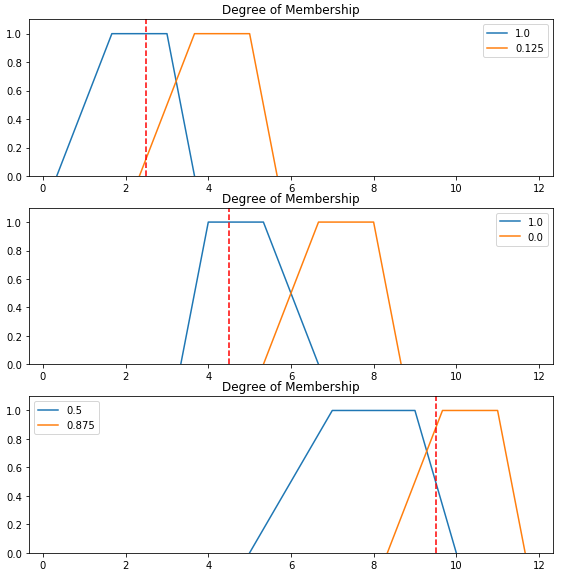
feaVec = np.array([2.5,4.5,9.5])

result = **CL\_FQRC\_Predict**(feaVec, fqmf, True)

print 'output:' + str(output)

Output:



result: [0.71428571 0.28571429]

### (d) Image Classification with Fuzzy Approach

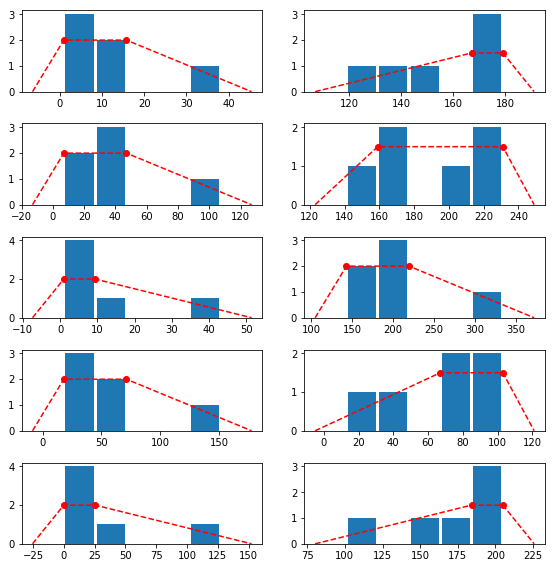
For Fuzzy classification, the FQRC [citation] is applied. The following function covered the

Sample code:

Output\_fuzzy = fcvt.Image\_Classification('Training', 'Testing', 'sift', ‘Fuzzy’)

Class: Forest

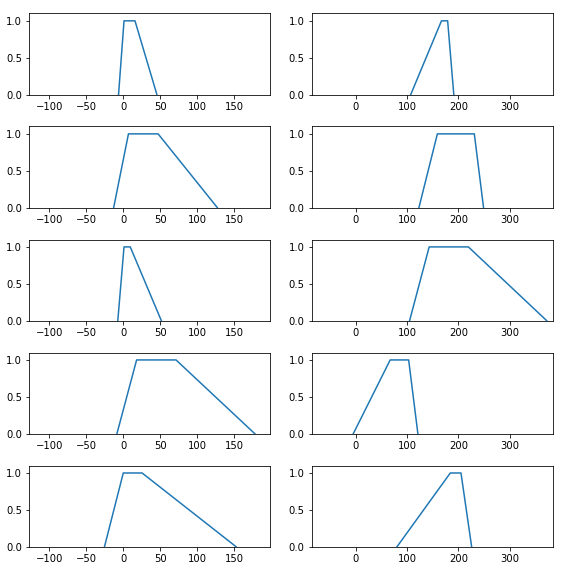
Class: Coast



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.

Class: Forest

Class: Coast



Feature 5

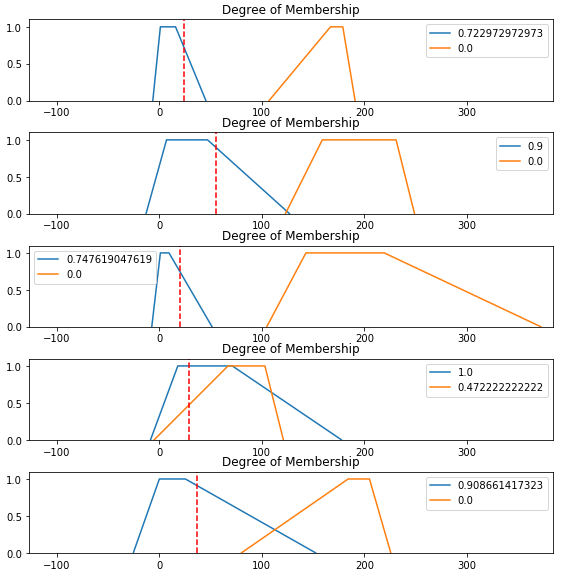
Feature 4

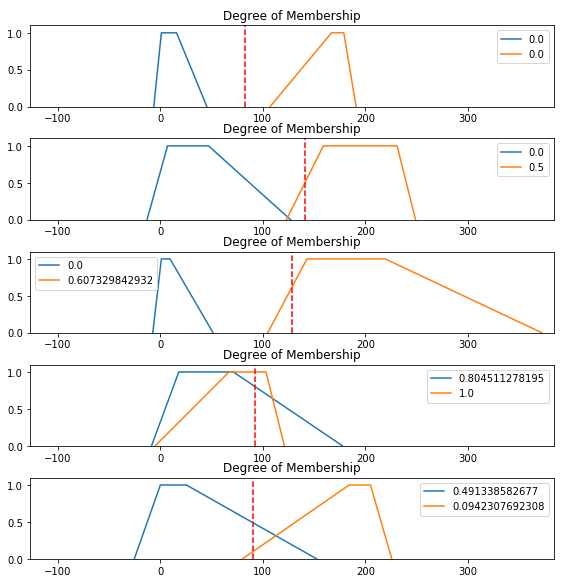
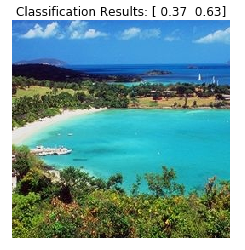
Feature 3

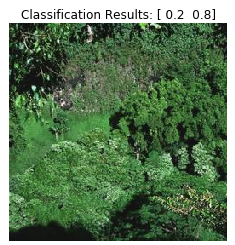
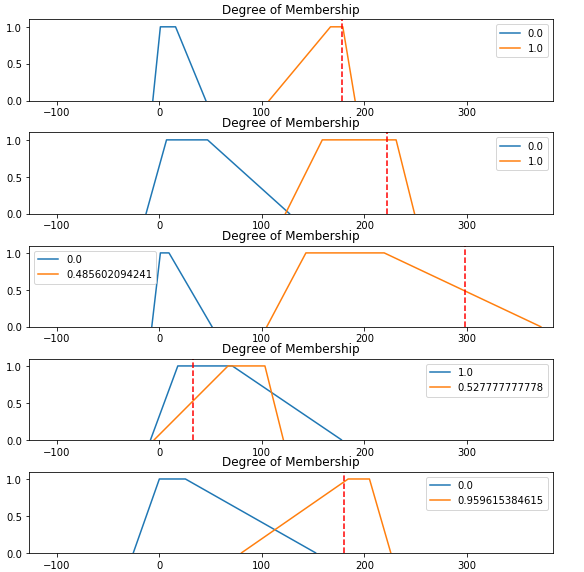
Feature 2

Feature 1

Output:





### (c) Limitation

Very depends on the features.

# **Extra(1): Fuzzy Deep Learning**

Introduce a little bit on Wei Ren works.

# **Extra(2): Read video**

1. Beforehand, please make sure you have “opencv\_ffmpeg2412.dll” in your local directory together with your code and sample of videos.
2. Read video from file and loop it for visualization. “*ret*” is true only if the video is successfully loaded and each frame of the video will be assigned to the variable “*frame*” for each loop.

cap = cv2.VideoCapture('walk.avi')

while(True):

ret, frame = cap.read()

if ret == True:

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

cv2.imshow('frame',frame)

if cv2.waitKey(1) & 0xFF == ord('q'):

break

else:

break

1. Perform background subtraction on the video.

cap = cv2.VideoCapture('walk.avi')

fgbg = cv2.BackgroundSubtractorMOG()

while(True):

ret, frame = cap.read()#

if ret == True:

fgmask = fgbg.apply(frame)

cv2.imshow('frame',fgmask)

if cv2.waitKey(1) & 0xFF == ord('q'):

break

else:

break

1. Perform motion tracking by using optical flow on the video.

cap = cv2.VideoCapture("walk.avi")

ret, frame1 = cap.read()

prvs = cv2.cvtColor(frame1,cv2.COLOR\_BGR2GRAY)

hsv = np.zeros\_like(frame1)

hsv[...,1] = 255

while(True):

ret, frame2 = cap.read()

if ret == True:

next = cv2.cvtColor(frame2,cv2.COLOR\_BGR2GRAY)

flow = cv2.calcOpticalFlowFarneback(prvs,next, 0.5, 3, 15, 3, 5, 1.2, 0)

mag, ang = cv2.cartToPolar(flow[...,0], flow[...,1])

hsv[...,0] = ang\*180/np.pi/2

hsv[...,2] = cv2.normalize(mag,None,0,255,cv2.NORM\_MINMAX)

rgb = cv2.cvtColor(hsv,cv2.COLOR\_HSV2BGR)

cv2.imshow('frame2',rgb)

k = cv2.waitKey(1) & 0xff

if k == ord('q'):

break

prvs = next

else:

break

1. Similar to image processing, we need to hold the visualization by using “cv2.waitKey(0)” until the user press any key for further processing or release the video capture with “*cap.release()*”. User may close all windows to release the memory by using “cv2.destroyAllWindows()”.

cv2.waitKey(0)

cap.release()

cv2.destroyAllWindows()

# **References:**

[1] 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.

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

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

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

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

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

<http://opencv.org/>

<http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_tutorials.html>

Credit

Introduction Page

Visualisation of membership generation (data driven)