This blog post illustrates the new functionality of the OpenImageR package (*Gabor Feature Extraction*). The Gabor features have been used extensively in [image analysis and processing](https://en.wikipedia.org/wiki/Gabor_filter) (Character and Face recognition). [Gabor](https://en.wikipedia.org/wiki/Dennis_Gabor) (Nobel prize winner, an electrical engineer, and physicist) used the following wording which I think it’s worth mentioning in this blog post, *“You can’t predict the future, but you can invent it.”* ([source](http://www.it.lut.fi/project/gabor/)).

In the following lines I’ll describe the *GaborFeatureExtract* R6 class which includes the following methods,

| **GaborFeatureExtract** |
| --- |
| gabor\_filter\_bank() |
| gabor\_feature\_extraction() |
| gabor\_feature\_engine() |
| plot\_gabor() |

These methods are based on the Matlab code [*Gabor Feature Extraction*](https://github.com/mhaghighat/gabor) of the paper *M. Haghighat, S. Zonouz, M. Abdel-Mottaleb, “CloudID: Trustworthy cloud-based and cross-enterprise biometric identification,” Expert Systems with Applications, vol. 42, no. 21, pp. 7905-7916, 2015*, http://dx.doi.org/10.1016/j.eswa.2015.06.025. The initial Matlab code was modified (I added the **Magnitude** feature and the **gabor\_feature\_engine()** method) and parallelized using Rcpp wherever it was possible.

**Gabor Features**

I came across the Gabor Features last month when I had to process images and I needed an additional function besides the already existing [HoG features](https://www.rdocumentation.org/packages/OpenImageR/versions/1.0.8/topics/HOG?tap_a=5644-dce66f&tap_s=10907-287229). I did the regular search on CRAN (Comprehensive R Archive Network) but I couldn’t find anything related to Gabor Feature Extraction (as of August 2018), therefore I decided to port the Matlab code into R.  
There are many resources on the web if someone intends to deepen his / her knowledge on the subject and I’ll add some of these that I found useful at the end of the blog post (References). I’ll explain the methods of the *GaborFeatureExtract* R6 class using an image found on a [Stackoverflow question](https://stackoverflow.com/a/20610474/8302386).

**gabor\_filter\_bank**

The **gabor\_filter\_bank** method *“… generates a custom-sized Gabor filter bank. It creates a UxV cell array, whose elements are MxN matrices; each matrix being a 2-D Gabor filter”* according to the [author of the Matlab](https://github.com/mhaghighat/gabor) code. The following code chunk shows how this works in R,

library(OpenImageR)

init\_gb = GaborFeatureExtract$new()

#------------------

# gabor-filter-bank

#------------------

gb\_f = init\_gb$gabor\_filter\_bank(scales = 5, orientations = 8, gabor\_rows = 39,

gabor\_columns = 39, plot\_data = TRUE)

#-----------------------------------------------

# plot of the real part of the gabor-filter-bank

#-----------------------------------------------

plt\_f = init\_gb$plot\_gabor(real\_matrices = gb\_f$gabor\_real, margin\_btw\_plots = 0.65,

thresholding = FALSE)

For the *gabor\_filter\_bank* I use 5 *scales* and 8 *orientations* to build filters of size 39 x 39. The output of the method is a list of length 3,

str(gb\_f)

List of 3

$ gaborArray :List of 40

..$ : cplx [1:39, 1:39] -1.58e-12-0.00i 0.00-5.02e-12i 1.50e-11-0.00i ...

..$ : cplx [1:39, 1:39] 4.86e-08-3.96e-08i 1.02e-07+4.63e-08i 6.31e-09+1.93e-07i ...

..$ : cplx [1:39, 1:39] 6.24e-06-6.24e-06i 1.18e-05+0.00i 1.10e-05+1.10e-05i ...

..$ : cplx [1:39, 1:39] -6.69e-05-3.18e-05i -4.63e-05-7.20e-05i -1.60e-06-9.81e-05i ...

..$ : cplx [1:39, 1:39] 1.40e-04+5.81e-05i 1.15e-04+1.15e-04i 6.68e-05+1.61e-04i ...

.......

$ gabor\_imaginary:List of 40

..$ : num [1:39, 1:39] -4.65e-27 -5.02e-12 -1.10e-26 4.21e-11 -2.99e-25 ...

..$ : num [1:39, 1:39] -3.96e-08 4.63e-08 1.93e-07 1.53e-07 -3.04e-07 ...

..$ : num [1:39, 1:39] -6.24e-06 4.84e-20 1.10e-05 2.01e-05 1.81e-05 ...

..$ : num [1:39, 1:39] -3.18e-05 -7.20e-05 -9.81e-05 -9.58e-05 -5.78e-05 ...

..$ : num [1:39, 1:39] 5.81e-05 1.15e-04 1.61e-04 1.86e-04 1.83e-04 ...

.......

$ gabor\_real :List of 40

..$ : num [1:39, 1:39] -1.58e-12 5.54e-27 1.50e-11 -4.12e-26 -1.11e-10 ...

..$ : num [1:39, 1:39] 4.86e-08 1.02e-07 6.31e-09 -2.85e-07 -4.28e-07 ...

..$ : num [1:39, 1:39] 6.24e-06 1.18e-05 1.10e-05 -8.11e-20 -1.81e-05 ...

..$ : num [1:39, 1:39] -6.69e-05 -4.63e-05 -1.60e-06 5.73e-05 1.12e-04 ...

..$ : num [1:39, 1:39] 1.40e-04 1.15e-04 6.68e-05 -3.77e-19 -7.57e-05 ...

.......

The first sublist (*gaborArray*) consists of 40 matrices (5 scales x 8 orientations) of type complex, where each matix is of dimension 39 x 39 (gabor filter). The second sublist (*gabor\_imaginary*) is the imaginary part (numeric) whereas the third sublist is the real part (*gabor\_real*). The real part (numeric) is used to plot the gabor filters.

The documentation includes more details for each of the parameters used.

**gabor\_feature\_extraction**

The **gabor\_feature\_extraction** method extracts the Gabor Features of the image. This method is modified in comparison to the initial Matlab code to give users the option to downsample the image or to normalize the features. Moreover, I added the **Magnitude** feature because according to [literature](https://www.researchgate.net/publication/283539985_Gabor_Feature_Based_Unsupervised_Change_Detection_of_Multitemporal_SAR_Images_Based_on_Two-Level_Clustering) it improves predictability.

Based on the previously mentioned *car.png* image,

# read image

#-----------

pth\_im = system.file("tmp\_images", "car.png", package = "OpenImageR")

im = readImage(pth\_im) \* 255

# gabor-feature-extract

#----------------------

gb\_im = init\_gb$gabor\_feature\_extraction(image = im, scales = 5, orientations = 8,

downsample\_gabor = FALSE, downsample\_rows = NULL,

downsample\_cols = NULL, gabor\_rows = 39,

gabor\_columns = 39, plot\_data = TRUE,

normalize\_features = FALSE, threads = 3)

This function again returns a list of length 3,

str(gb\_im)

List of 3

$ gaborFeatures :List of 2

..$ magnitude : num [1, 1:206670] 0 0 0 0 0 0 0 0 0 0 ...

..$ energy\_aptitude: num [1, 1:80] 115823 27566 33289 20074 26918 ...

$ gabor\_features\_imaginary:List of 5

..$ : num [1:166, 1:249, 1:8] 0 0 0 0 0 0 0 0 0 0 ...

..$ : num [1:166, 1:249, 1:8] 0 0 0 0 0 0 0 0 0 0 ...

..$ : num [1:166, 1:249, 1:8] 0 0 0 0 0 0 0 0 0 0 ...

..$ : num [1:166, 1:249, 1:8] 0 0 0 0 0 0 0 0 0 0 ...

..$ : num [1:166, 1:249, 1:8] 0 0 0 0 0 0 0 0 0 0 ...

$ gabor\_features\_real :List of 5

..$ : num [1:166, 1:249, 1:8] 0 0 0 0 0 0 0 0 0 0 ...

..$ : num [1:166, 1:249, 1:8] 0 0 0 0 0 0 0 0 0 0 ...

..$ : num [1:166, 1:249, 1:8] 0 0 0 0 0 0 0 0 0 0 ...

..$ : num [1:166, 1:249, 1:8] 0 0 0 0 0 0 0 0 0 0 ...

..$ : num [1:166, 1:249, 1:8] 0 0 0 0 0 0 0 0 0 0 ...

where *gaborFeatures* are the extracted features with number of rows equal to nrow(im) x ncol(im) (or 166 x 249) and number of columns equal to scales x orientations (or 5 x 8). The second and third sublists are the imaginary and real part of the image resulted after the convolution with the gabor filters. The following code chunk allows the user to plot the different *scales* and *orientations* of the image,

plt\_im = init\_gb$plot\_gabor(real\_matrices = gb\_im$gabor\_features\_real,

margin\_btw\_plots = 0.65, thresholding = FALSE)

By thresholding the *gb\_im$gabor\_features\_real* object (scales, orientations) to [0,1] the images can be visually explored,

# thresholding parameter is set to TRUE

#--------------------------------------

plt\_im\_thresh = init\_gb$plot\_gabor(real\_matrices = gb\_im$gabor\_features\_real,

margin\_btw\_plots = 0.65, thresholding = TRUE)

**gabor\_feature\_engine**

The *gabor\_feature\_engine* method is an extension of the initial Matlab code and allows the user to extract gabor features from multiple images. This method works in the same way as the *HOG\_apply* method, which takes a matrix of images – such as the mnist data set – and after processing it returns the features. The following example illustrates how to use the *gabor\_feature\_engine* method with the mnist data set,

# for instance downloading of the mnist data on a linux OS

#----------------------------------------------------------

system("wget https://raw.githubusercontent.com/mlampros/DataSets/master/mnist.zip")

# or just navigate to "https://github.com/mlampros/DataSets/blob/master/mnist.zip" and click the download button

#-------------------------------------------------------------------------------------------

mnist <- read.table(unz("mnist.zip", "mnist.csv"), nrows = 70000, header = T,

quote = "\"", sep = ",")

mnist = as.matrix(mnist)

x = mnist[, -ncol(mnist)]

y = mnist[, ncol(mnist)] + 1

dat = init\_gb$gabor\_feature\_engine(img\_data = x, img\_nrow = 28, img\_ncol = 28,

scales = 4, orientations = 8, gabor\_rows = 13,

gabor\_columns = 13, downsample\_gabor = FALSE,

downsample\_rows = NULL, downsample\_cols = NULL,

normalize\_features = FALSE, threads = 6,

verbose = TRUE)

Time to complete : 4.111672 mins

> str(dat)

List of 2

$ magnitude : num [1:70000, 1:3136] 0 0 0 0 0 0 0 0 0 0 ...

$ energy\_aptitude: num [1:70000, 1:64] 2682 2576 1399 1178 2240 ...

The *dat* object is a list of length 2. The first sublist corresponds to *magnitude* whereas the second sublist to *local-energy* and *mean-aptitude*. The first thing to do before calculating the accuracy for the mnist data is to reduce the dimensionality of the *magnitude* feature (I’ll use the [irlba](https://github.com/bwlewis/irlba) package for this purpose),

svd\_irlb = irlba::irlba(as.matrix(dat$magnitude), nv = 100, nu = 100, verbose = T)

new\_x = as.matrix(dat$magnitude) %\*% svd\_irlb$v

and I’ll create a centered-scaled matrix of the reduced *magnitude* and the *energy-aptitude* data,

dat = ClusterR::center\_scale(cbind(new\_x, dat$energy\_aptitude))

dim(dat)

[1] 70000 164

then I’ll utilize the *nmslibR* library (approximate method ‘hnsw’) to calculate the accuracy for the mnist data,

M = 30

efC = 100

num\_threads = 6

index\_params = list('M'= M, 'indexThreadQty' = num\_threads, 'efConstruction' = efC,

'post' = 0, 'skip\_optimized\_index' = 1 )

efS = 100

query\_time\_params = list('efSearch' = efS)

fit\_gab = nmslibR::KernelKnnCV\_nmslib(dat, y, k = 20, folds = 4, h = 1,

weights\_function = 'biweight\_triangular\_tricube\_MULT',

Levels = sort(unique(y)), Index\_Params = index\_params,

Time\_Params = query\_time\_params, space = "cosinesimil",

space\_params = NULL, method = "hnsw",

data\_type = "DENSE\_VECTOR", dtype = "FLOAT",

index\_filepath = NULL, print\_progress = T,

num\_threads = 6, seed\_num = 1)

time to complete : 30.82252 secs

acc\_fit\_gab = unlist(lapply(1:length(fit\_gab$preds),

function(x) {

mean(y[fit\_gab$folds[[x]]] == max.col(

fit\_gab$preds[[x]], ties.method = 'random')) }))

mean(acc\_fit\_gab)

[1] 0.9752714

I’ll do the same using the *HOG\_apply* function,

# hog-features for the mnist data

#--------------------------------

hog = OpenImageR::HOG\_apply(x, cells = 6, orientations = 9, rows = 28, columns = 28, threads = 6)

fit\_hog = nmslibR::KernelKnnCV\_nmslib(hog, y, k = 20, folds = 4, h = 1,

weights\_function = 'biweight\_triangular\_tricube\_MULT',

Levels = sort(unique(y)), Index\_Params = index\_params,

Time\_Params = query\_time\_params, space = "cosinesimil",

space\_params = NULL, method = "hnsw",

data\_type = "DENSE\_VECTOR", dtype = "FLOAT",

index\_filepath = NULL, print\_progress = T,

num\_threads = 6, seed\_num = 1)

time to complete : 41.00309 secs

acc\_fit\_hog = unlist(lapply(1:length(fit\_hog$preds),

function(x) {

mean(y[fit\_hog$folds[[x]]] == max.col(

fit\_hog$preds[[x]], ties.method = 'random')) }))

mean(acc\_fit\_hog)

[1] 0.9787429

By averaging both the *gabor* and the *HoG* features the mean accuracy increases to 98.34 % which is very close to the score of the [HoG + brute force method of KernelKnn](http://mlampros.github.io/2016/07/10/KernelKnn/) (98.4),

acc\_fit\_gab\_hog = unlist(lapply(1:length(fit\_gab$preds),

function(x) {

mean(y[fit\_gab$folds[[x]]] == max.col(

(fit\_gab$preds[[x]] + fit\_hog$preds[[x]]) / 2,

ties.method = 'random')) }))

mean(acc\_fit\_gab\_hog)

[1] 0.9833857