

```
1 from scipy import misc
2 from PIL import Image
3 from skimage import exposure
4 from sklearn import svm
5
6 import scipy
7 from math import sqrt, pi
8 from numpy import exp
9 from matplotlib import pyplot as plt
10 import numpy as np
11 import glob
12 import matplotlib.pyplot as pltss
13 import cv2
14 from matplotlib import cm
15 import pandas as pd
16 from math import pi, sqrt
17 import pywt
18
19 #img_rows=img_cols=200
20 immatrix=[]
21 im_unpre = []
22 #image_path = Image.open('C:\Users\Priyansh and
23 Ananya\Desktop\Diabetic_Retinopathy\diaretdb1_v_1_1\diaretdb1_v_1_1\resources
24 \images\ddb1_fundusimages\image0')
25 #image = misc.imread(image_path)
26
27 for i in range(1,90):
28     img_pt = r'C:\Users\ Priyansh and Ananya
29 \Desktop\Diabetic_Retinopathy\diaretdb1_v_1_1\diaretdb1_v_1_1\resources\image
30 s\ddb1_fundusimages\image'
31     if i < 10:
32         img_pt = img_pt + "00" + str(i) + ".png"
33     else:
34         img_pt = img_pt + "0" + str(i)+ ".png"
35
36     img = cv2.imread(img_pt)
37     #im_unpre.append(np.array(img).flatten())
38     img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
39     equ = cv2.equalizeHist(img_gray)
40     immatrix.append(np.array(equ).flatten())
41     #res = np.hstack((img_gray,equ))
42
43 np.shape(np.array(equ).flatten())
44 np.shape(immatrix)
45 np.shape(equ)
46 plt.imshow(immatrix[78].reshape((1152,1500)), cmap='gray')
47 plt.show()
48
49 imm_dwt = []
50 for equ in immatrix:
51     equ = equ.reshape((1152,1500))
52     coeffs = pywt.dwt2(equ, 'haar')
53     equ2 = pywt.idwt2(coeffs, 'haar')
54     imm_dwt.append(np.array(equ2).flatten())
55
```

```

52 np.shape(imm_dwt)
53 np.shape(equ2)
54 plt.imshow(imm_dwt[78].reshape((1152,1500)), cmap='gray')
55 plt.show()
56
57 def _filter_kernel_mf_fdog(L, sigma, t = 3, mf = True):
58     dim_y = int(L)
59     dim_x = 2 * int(t * sigma)
60     arr = np.zeros((dim_y, dim_x), 'f')
61
62     ctr_x = dim_x / 2
63     ctr_y = int(dim_y / 2.)
64
65     # an un-natural way to set elements of the array
66     # to their x coordinate.
67     # x's are actually columns, so the first dimension of the iterator is
used
68     it = np.nditer(arr, flags=['multi_index'])
69     while not it.finished:
70         arr[it.multi_index] = it.multi_index[1] - ctr_x
71         it.iternext()
72
73     two_sigma_sq = 2 * sigma * sigma
74     sqrt_w_pi_sigma = 1. / (sqrt(2 * pi) * sigma)
75     if not mf:
76         sqrt_w_pi_sigma = sqrt_w_pi_sigma / sigma ** 2
77
78     #@vectorize(['float32(float32)'], target='cpu')
79     def k_fun(x):
80         return sqrt_w_pi_sigma * exp(-x * x / two_sigma_sq)
81
82     #@vectorize(['float32(float32)'], target='cpu')
83     def k_fun_derivative(x):
84         return -x * sqrt_w_pi_sigma * exp(-x * x / two_sigma_sq)
85
86     if mf:
87         kernel = k_fun(arr)
88         kernel = kernel - kernel.mean()
89     else:
90         kernel = k_fun_derivative(arr)
91
92     # return the "convolution" kernel for filter2D
93     return cv2.flip(kernel, -1)
94
95 def show_images(images, titles=None, scale=1.3):
96     """Display a list of images"""
97     n_ims = len(images)
98     if titles is None: titles = ['(%d)' % i for i in range(1, n_ims + 1)]
99     fig = plt.figure()
100     n = 1
101     for image, title in zip(images, titles):
102         a = fig.add_subplot(1, n_ims, n) # Make subplot
103         if image.ndim == 2: # Is image grayscale?
104             plt.imshow(image, cmap = cm.Greys_r)
105         else:

```

```

105         plt.imshow(cv2.cvtColor(image, cv2.COLOR_RGB2BGR))
106         a.set_title(title)
107         plt.axis("off")
108         n += 1
109     fig.set_size_inches(np.array(fig.get_size_inches(), dtype=np.float) *
110 n_ims / scale)
111     plt.show()
112
113
114 def gaussian_matched_filter_kernel(L, sigma, t = 3):
115     '''
116     K = 1/(sqrt(2 * pi) * sigma ) * exp(-x^2/2sigma^2), |y| <= L/2, |x| < s
117     * t
118     '''
119     return _filter_kernel_mf_fdog(L, sigma, t, True)
120
121 #Creating a matched filter bank using the kernel generated from the above
122 #functions
123 def createMatchedFilterBank(K, n = 12):
124     rotate = 180 / n
125     center = (K.shape[1] / 2, K.shape[0] / 2)
126     cur_rot = 0
127     kernels = [K]
128
129     for i in range(1, n):
130         cur_rot += rotate
131         r_mat = cv2.getRotationMatrix2D(center, cur_rot, 1)
132         k = cv2.warpAffine(K, r_mat, (K.shape[1], K.shape[0]))
133         kernels.append(k)
134
135     return kernels
136
137 #Given a filter bank, apply them and record maximum response
138
139 def applyFilters(im, kernels):
140     images = np.array([cv2.filter2D(im, -1, k) for k in kernels])
141     return np.max(images, 0)
142
143 gf = gaussian_matched_filter_kernel(20, 5)
144 bank_gf = createMatchedFilterBank(gf, 4)
145
146 imm_gauss = []
147 for equ2 in imm_dwt:
148     equ2 = equ2.reshape((1152, 1500))
149     equ3 = applyFilters(equ2, bank_gf)
150     imm_gauss.append(np.array(equ3).flatten())
151
152 np.shape(imm_gauss)
153 plt.imshow(imm_gauss[78].reshape((1152, 1500)), cmap='gray')
154 plt.show()
155
156 def createMatchedFilterBank():
157     filters = []

```

```

157     filters = []
158     ksize = 31
159     for theta in np.arange(0, np.pi, np.pi / 16):
160         kern = cv2.getGaborKernel((ksize, ksize), 6, theta, 12, 0.37, 0,
ktype=cv2.CV_32F)
161         kern /= 1.5*kern.sum()
162         filters.append(kern)
163     return filters
164
165 def applyFilters(im, kernels):
166     images = np.array([cv2.filter2D(im, -1, k) for k in kernels])
167     return np.max(images, 0)
168
169 bank_gf = createMatchedFilterBank()
170 #equx=equ3
171 #equ3 = applyFilters(equ2,bank_gf)
172 imm_gauss2 = []
173 for equ2 in imm_dwt:
174     equ2 = equ2.reshape((1152,1500))
175     equ3 = applyFilters(equ2,bank_gf)
176     imm_gauss2.append(np.array(equ3).flatten())
177
178 np.shape(imm_gauss2)
179 plt.imshow(imm_gauss2[20].reshape((1152,1500)), cmap='gray')
180 plt.show()
181
182 np.shape(imm_gauss2)
183 plt.imshow(imm_gauss2[1].reshape((1152,1500)), cmap='gray')
184 plt.show()
185
186 e_ = equ3
187 np.shape(e_)
188 e_=e_.reshape((-1,3))
189 np.shape(e_)
190
191 img = equ3
192 Z = img.reshape((-1,3))
193
194 # convert to np.float32
195 Z = np.float32(Z)
196
197 k=cv2.KMEANS_PP_CENTERS
198
199
200 # define criteria, number of clusters(K) and apply kmeans()
201 criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)
202 K = 2
203 ret,label,center=cv2.kmeans(Z,K,None,criteria,10,k)
204
205 # Now convert back into uint8, and make original image
206 center = np.uint8(center)
207 res = center[label.flatten()]
208 res2 = res.reshape((img.shape))
209
210 imm_kmean = []
211 for equ2 in imm_gauss2:

```

```
211 for equ3 in imm_gauss2:
212     img = equ3.reshape((1152,1500))
213     Z = img.reshape((-1,3))
214
215     # convert to np.float32
216     Z = np.float32(Z)
217
218     k=cv2.KMEANS_PP_CENTERS
219
220
221     # define criteria, number of clusters(K) and apply kmeans()
222     criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)
223     K = 2
224     ret,label,center=cv2.kmeans(Z,K,None,criteria,10,k)
225
226     # Now convert back into uint8, and make original image
227     center = np.uint8(center)
228     res = center[label.flatten()]
229     res2 = res.reshape((img.shape))
230     imm_kmean.append(np.array(res2).flatten())
231
232 np.shape(imm_kmean)
233 plt.imshow(imm_kmean[78].reshape((1152,1500)), cmap="gray")
234 plt.show()
235
236 from sklearn.svm import SVC
237 clf = SVC()
238 Y = np.ones(89)
239 Y[1]=Y[5]=Y[7]=Y[17]=Y[6]=0
240
241 clf.fit(imm_kmean, Y)
242 y_pred = clf.predict(imm_kmean)
243
244 k =
    [1,3,4,9,10,11,13,14,20,22,24,25,26,27,28,29,35,36,38,42,53,55,57,64,70,79,84
    ,86]
245 k = k-np.ones(len(k))
246 k=[int(x) for x in k]
247
248 imm_train = []
249 y_train = []
250 k.append(5)
251 k.append(7)
252 for i in k:
253     imm_train.append(imm_kmean[i])
254     y_train.append(Y[i])
255
256 y_pred = clf.predict(imm_kmean)
257
258
259 from sklearn.neighbors import KNeighborsClassifier
260 neigh = KNeighborsClassifier(n_neighbors=3)
261 neigh.fit(imm_train, y_train)
262 y_pred2=neigh.predict(imm_kmean)
263 neigh.score(imm_kmean,Y)
```