

```

1 from __future__ import division, print_function
2 import argparse
3 import matplotlib.image as mpimg
4 import matplotlib.pyplot as plt
5 import numpy as np
6 import os
7 import random
8
9
10 def init_centroids(num_clusters, image):
11     """
12     Initialize a `num_clusters` x image_shape[-1] nparray to RGB
13     values of randomly chosen pixels of `image`
14
15     Parameters
16     -----
17     num_clusters : int
18         Number of centroids/clusters
19     image : nparray
20         (H, W, C) image represented as an nparray
21
22     Returns
23     -----
24     centroids_init : nparray
25         Randomly initialized centroids
26     """
27
28     # *** START YOUR CODE ***
29     # raise NotImplementedError('init_centroids function not implemented')
30     H, W, C = np.shape(image)
31     centroids_init = np.zeros(shape=[num_clusters, C])
32
33     for idx in range(num_clusters):
34         h, w = random.randint(1, H - 1), random.randint(1, W - 1)
35         centroids_init[idx, :] = image[h, w, :]
36     # *** END YOUR CODE ***
37
38     return centroids_init
39
40
41 def update_centroids(centroids, image, max_iter=30, print_every=10):
42     """
43     Carry out k-means centroid update step `max_iter` times
44
45     Parameters
46     -----
47     centroids : nparray
48         The centroids stored as an nparray
49     image : nparray
50         (H, W, C) image represented as an nparray
51     max_iter : int
52         Number of iterations to run
53     print_every : int
54         Frequency of status update
55
56     Returns
57     -----
58     new_centroids : nparray
59         Updated centroids
60     """
61
62     # *** START YOUR CODE ***
63     # raise NotImplementedError('update_centroids function not implemented')
64     H, W, C = np.shape(image)
65     num_clusters = len(centroids)
66     new_centroids = np.zeros(shape=[num_clusters, C])
67
68     for it in range(max_iter):
69         # Usually expected to converge long before `max_iter` iterations
70         if it == 0 or (it + 1) % print_every == 0:
71             print("[INFO] Completed iteration {} of {}".format(it + 1, max_iter))
72             new_centroids = np.zeros(shape=[num_clusters, C])

```

```

73     new_assignments = np.zeros(shape=[num_clusters, 1])
74
75     for x in range(H):
76
77         for y in range(W):
78             # Initialize `dist` vector to keep track of distance to every centroid
79             dist = np.zeros(shape=[num_clusters, 1])
80
81             # Loop over all centroids and store distances in `dist`
82             for idx in range(num_clusters):
83                 d = centroids[idx, :] - image[x, y, :]
84                 dist[idx] = np.dot(np.transpose(d), d)
85
86             # Find closest centroid and update `new_centroids`
87             centroid_idx = dist.argmin()
88             new_assignments[centroid_idx] += 1
89             new_centroids[centroid_idx, :] += image[x, y, :]
90
91         # Update `new_centroids`
92         for idx in range(num_clusters):
93             if new_assignments[idx] > 0:
94                 new_centroids[idx, :] = new_centroids[idx, :] / new_assignments[idx]
95     # *** END YOUR CODE ***
96
97     return new_centroids
98
99

```

```

100 def update_image(image, centroids):

```

```

101     """
102     Update RGB values of pixels in `image` by finding
103     the closest among the `centroids`

```

```

104
105     Parameters

```

```

106     -----

```

```

107     image : nparray

```

```

108         (H, W, C) image represented as an nparray

```

```

109     centroids : int

```

```

110         The centroids stored as an nparray
111

```

```

112     Returns

```

```

113     -----

```

```

114     image : nparray

```

```

115         Updated image
116     """

```

```

117
118     # *** START YOUR CODE ***

```

```

119     # raise NotImplementedError('update_image function not implemented')

```

```

120     H, W, C = np.shape(image)

```

```

121     num_clusters = len(centroids)

```

```

122
123     for x in range(H):

```

```

124
125         for y in range(W):

```

```

126             # Initialize `dist` vector to keep track of distance to every centroid
127             dist = np.zeros(shape=[num_clusters, 1])

```

```

128
129             # Loop over all centroids and store distances in `dist`

```

```

130             for idx in range(num_clusters):

```

```

131                 d = centroids[idx, :] - image[x, y, :]

```

```

132                 dist[idx] = np.dot(np.transpose(d), d)

```

```

133
134             # Find closest centroid and update pixel value in `image`

```

```

135             centroid_idx = dist.argmin()

```

```

136             image[x, y, :] = centroids[centroid_idx]

```

```

137     # *** END YOUR CODE ***

```

```

138
139     return image

```

```

140
141
142 def main(args):

```

```

143
144     # Setup

```



```

145 max_iter = args.max_iter
146 print_every = args.print_every
147 image_path_small = args.small_path
148 image_path_large = args.large_path
149 num_clusters = args.num_clusters
150 figure_idx = 0
151
152 # Load small image
153 image = np.copy(mping.imread(image_path_small))
154 print('[INFO] Loaded small image with shape: {}'.format(np.shape(image)))
155 plt.figure(figure_idx)
156 figure_idx += 1
157 plt.imshow(image)
158 plt.title('Original small image')
159 plt.axis('off')
160 savepath = os.path.join('.', 'orig_small.png')
161 plt.savefig(savepath, transparent=True, format='png', bbox_inches='tight')
162
163 # Initialize centroids
164 print('[INFO] Centroids initialized')
165 centroids_init = init_centroids(num_clusters, image)
166
167 # Update centroids
168 print(25 * '=')
169 print('Updating centroids ...')
170 print(25 * '=')
171 centroids = update_centroids(centroids_init, image, max_iter, print_every)
172
173 # Load large image
174 image = np.copy(mping.imread(image_path_large))
175 image.setflags(write=1)
176 print('[INFO] Loaded large image with shape: {}'.format(np.shape(image)))
177 plt.figure(figure_idx)
178 figure_idx += 1
179 plt.imshow(image)
180 plt.title('Original large image')
181 plt.axis('off')
182 savepath = os.path.join('.', 'orig_large.png')
183 plt.savefig(fname=savepath, transparent=True, format='png', bbox_inches='tight')
184
185 # Update large image with centroids calculated on small image
186 print(25 * '=')
187 print('Updating large image ...')
188 print(25 * '=')
189 image_clustered = update_image(image, centroids)
190
191 plt.figure(figure_idx)
192 figure_idx += 1
193 plt.imshow(image_clustered)
194 plt.title('Updated large image')
195 plt.axis('off')
196 savepath = os.path.join('.', 'updated_large.png')
197 plt.savefig(fname=savepath, transparent=True, format='png', bbox_inches='tight')
198
199 print('\nCOMPLETE')
200 plt.show()
201
202
203 if __name__ == '__main__':
204     parser = argparse.ArgumentParser()
205     parser.add_argument('--small_path', default='./peppers-small.tiff',
206                         help='Path to small image')
207     parser.add_argument('--large_path', default='./peppers-large.tiff',
208                         help='Path to large image')
209     parser.add_argument('--max_iter', type=int, default=150,
210                         help='Maximum number of iterations')
211     parser.add_argument('--num_clusters', type=int, default=16,
212                         help='Number of centroids/clusters')
213     parser.add_argument('--print_every', type=int, default=10,
214                         help='Iteration print frequency')
215     args = parser.parse_args()
216     main(args)

```

