


<https://swayam.gov.in>

https://swayam.gov.in/nc_details/NPTEL

reviewer5@nptel.iitm.ac.in

[NPTEL \(https://swayam.gov.in/explorer?ncCode=NPTEL\)](https://swayam.gov.in/explorer?ncCode=NPTEL) » [Image Signal Processing \(course\)](#)
[Announcements \(announcements\)](#)
[About the Course \(preview\)](#)
[Ask a Question \(forum\)](#)
[Progress \(student/home\)](#)
[Mentor \(student/mentor\)](#)

Programming assignment: Scale an image

Due on 2020-10-01, 23:59 IST

It is recommended to initially work on this assignment using Google Colaboratory ("Colab" for short is a free Jupyter notebook environment provided by Google that allows you to run Python in your browser). The introduction videos for Colab will be shared in discussion forum. Being said that, this is a recommended way to do the assignments. You can always directly work on NPTEL website.

Follow these instructions to work on the assignment in google-colab. Click on this Assignment-1 file (<https://colab.research.google.com/drive/1tSscLDV9PlmBwfEPjoi81e9YOUyqz4-b?usp=sharing>). Make a copy of it in your drive. Right click and open the file using Google Colaboratory (You first need to log in to your google account). When you're ready to verify/submit your assignment, paste the missing code snippets.

In this assignment you will perform scaling on an image. The image has already been loaded for you, in the variable named 'cells'. Your task is to scale this image by 0.8 (both horizontally and vertically).

Instructions:

1. You are required to fill in the missing details. The places where you are expected to supply code begin and end with `# <---` and `# --->` respectively.
2. Please read the comments carefully to understand what is being asked of you.
3. Make sure that you always do Target-Source (T-S) mapping.

Private Test cases used for evaluation

Test Case 1

Input

cells image

Expected Output

ok

Actual Output

ok

Status

Passed

The due date for submitting this assignment has passed.

1 out of 1 tests passed.

You scored 100.0/100.

Assignment submitted on 2020-10-06, 09:44 IST

Your last recorded submission was :

```

1 import base64
2 import io
3
4 cells_scale_str = b'ivB0Rw0KGgoAAAANSUHEUgAAATsAAADwCAAAAABa75zYAAAB3RJTUUH5AYIERgF5GCL6gAAIABJREFUeJxMumeQJWd2
5
6 image_bytes = base64.b64decode(cells_scale_str)
7
8 image_file = io.BytesIO(image_bytes)
9 import numpy as np
10 import imageio
11
12 def image_size(image):
13     if image.ndim == 2:
14         return image.shape
15     else:
16         return image.shape[:-1]
17
18
19 def bilinear_interpolation(source_image, source_point):
20     """
21     Computes the intensity at `source_point` by bilinearly interpolating
22     intensities in the immediate 2 X 2 neighborhood of the `source_point`.
23
24     Args:
25         source_image (np.array): The source image
26         source_point (float, float): The source point
27
28     Returns:
29         uint8: Pixel intensity at source_point
30     """
31     i_s, j_s = source_point
32
33     # Floor `i_s` to get `i`
34     i = int(np.floor(i_s))
35
36     # Similarly, compute `j`
37     # <---
38     j = int(np.floor(j_s))
39     # --->
40
41     # The co-ordinates of the top-left (`tl`) corner are simply (i, j)
42     tl = i, j
43
44     # Write down the co-ordinates of the remaining three corners
45     # (top-right, bottom-left, bottom-right) below.
46     # Use the variable names `tr`, `bl`, `br` respectively.
47
48     # <---
49     tr = i, j + 1
50     bl = i + 1, j

```

Course outline

How does an NPTEL online course work?

Week 1

- Course Introduction (unit? unit=17&lesson=29)
- Applications of Image processing (unit? unit=17&lesson=30)
- Applications of Image processing (continued) (unit? unit=17&lesson=31)
- Basics of Images (unit? unit=17&lesson=32)
- Shot Noise (unit? unit=17&lesson=33)
- Geometric Transformations (unit?unit=17&lesson=34)
- Geometric Transformations (continued) (unit? unit=17&lesson=35)
- Bilinear Interpolation (unit? unit=17&lesson=36)
- Programming assignment: Translate an image (/noc20_ee83/progassignment? name=113)
- Programming assignment: Rotate an image (/noc20_ee83/progassignment? name=115)
- Programming assignment: Scale an image (/noc20_ee83/progassignment? name=117)
- Image Signal Processing : Week 1 Feedback Form (unit?unit=17&lesson=146)
- Week 1 Assignments solutions (unit? unit=17&lesson=162)
- Lecture materials (unit? unit=17&lesson=165)

Week 2

Week 3

Week 4

Week 5

Week 6

Week 7

```

51     br = i + 1, j + 1
52     # --->
53
54     # Next, we compute the distance of `source_point` from top-left corner along
55     # vertical and horizontal directions separately.
56     del_i, del_j = i_s - i, j_s - j
57
58     # Create a variable called `pixel intensity` and assign the
59     # pixel value obtained by bilinearly interpolating pixel values
60     # at tl, tr, bl, br.
61     # Use `del_i`, `del_j` computed in the previous step to obtain
62     # the weights for interpolation.
63     # <---
64     pixel_intensity = (1 - del_i) * (1 - del_j) * source_image[tl] + \
65                     (1 - del_i) * del_j * source_image[tr] + \
66                     del_i * (1 - del_j) * source_image[bl] + \
67                     del_i * del_j * source_image[br]
68     # --->
69     return np.uint8(pixel_intensity)
70
71 def transform(source_image, transformation, target_size=None):
72     """
73     Transforms `source_image` as dictated by `transformation`.
74
75     Note that this function does T-S mapping. So, `transformation` is actually from Target to Source.
76
77     Args:
78         source_image (np.array): The source image
79         transformation (np.array): 3 x 3 transformation matrix
80         target_size (uint, uint): Size of the target_image
81
82     Returns:
83         np.array: Transformed image
84
85     source_rows, source_cols = image_size(source_image)
86
87     # When no `target_size` is supplied, `target_image` will be the same size as `source_image`
88     target_rows, target_cols = target_size if target_size else (source_rows, source_cols)
89     target_image = np.zeros((target_rows, target_cols), dtype=np.uint8)
90
91     # We iterate over each pixel in `target_image` and assign the appropriate intensity
92     for i_t in range(target_rows):
93         for j_t in range(target_cols):
94             # Map each target point `(i_t`, `j_t`) through `transformation`
95             # to obtain its corresponding source_point `(i_s`, `j_s`)
96             # <---
97             i_s, j_s, v = np.array([i_t, j_t, 1]) @ transformation.T
98             i_s, j_s = i_s / v, j_s / v
99             # --->
100
101             # We ignore all target points whose source points lie outside the
102             # source image. All these intensities remain 0.
103             if 0 <= i_s < source_rows - 1 and 0 <= j_s < source_cols - 1:
104                 # Assign the intensity value of target_image at `(i_t, j_t)` using the
105                 # bilinear interpolation function above.
106                 # <---
107                 target_image[i_t, j_t] = bilinear_interpolation(source_image, (i_s, j_s))
108                 # --->
109
110     return target_image
111
112 def scale(source_image, factor):
113     """
114     Scales the `source_image` by `factor` in both dimensions.
115
116     Args:
117         source_image (np.array): The source image
118         factor (float): Scaling factor.
119
120     Returns:
121         np.array: Scaled image
122
123     # Create a variable called `scaling` which holds a 3 x 3
124     # numpy array that corresponds to scaling by factor
125     # Note that this matrix should be T-S,
126     # so it would be the inverse of what you might think of for
127     # an S-T scaling matrix.
128     # <---
129     scaling = np.array([[1 / factor, 0, 0],
130                       [0, 1 / factor, 0],
131                       [0, 0, 1]])
132     # --->
133     return transform(source_image, scaling)
134
135 cells = imageio.imread(image_file, format='PNG')
136 # Call the `scale` function above with the right parameters.
137 # <---
138 cells_scaled = scale(cells, 0.8)
139 # --->
140
141 def bilinear_interpolation_c(source_image, source_point):
142     """
143     Computes the intensity at `source_point` by bilinearly interpolating
144     intensities in the immediate 2 X 2 neighborhood of the `source_point`.
145
146     Args:
147         source_image (np.array): The source image
148         source_point (float, float): The source point
149
150     Returns:
151         uint8: Pixel intensity at source_point
152
153         i_s, j_s = source_point
154
155         # Floor `i_s` to get `i`
156         i = int(np.floor(i_s))
157
158         # Similarly, compute `j`
159         # <---
160         j = int(np.floor(j_s))
161         # --->
162
163         # The co-ordinates of the top-left (`tl`) corner are simply (i, j)
164         tl = i, j

```

```

172 # Write down the co-ordinates of the remaining three corners
173 # (top-right, bottom-left, bottom-right) below.
174 # Use the variable names `tr`, `bl`, `br` respectively.
175
176 # <---
177 tr = i, j + 1
178 bl = i + 1, j
179 br = i + 1, j + 1
180 # --->
181
182 # Next, we compute the distance of `source_point` from top-left corner along
183 # vertical and horizontal directions separately.
184 del_i, del_j = i_s - i, j_s - j
185
186 # Create a variable called `pixel_intensity` and assign the
187 # pixel value obtained by bilinearly interpolating pixel values
188 # at tl, tr, bl, br.
189 # Use `del_i`, `del_j` computed in the previous step to obtain
190 # the weights for interpolation.
191 # <---
192 pixel_intensity = (1 - del_i) * (1 - del_j) * source_image[tl] + \
193                 (1 - del_i) * del_j * source_image[tr] + \
194                 del_i * (1 - del_j) * source_image[bl] + \
195                 del_i * del_j * source_image[br]
196 # --->
197 return np.uint8(pixel_intensity)
198
199 def transform_c(source_image, transformation, target_size=None):
200     """
201     Transforms `source_image` as dictated by `transformation`.
202
203     Note that this function does T-S mapping. So, `transformation` is actually from Target to Source.
204
205     Args:
206         source_image (np.array): The source image
207         transformation (np.array): 3 x 3 transformation matrix
208         target_size (uint, uint): Size of the target_image
209
210     Returns:
211         np.array: Transformed image
212     """
213     source_rows, source_cols = image_size(source_image)
214
215     # When no `target_size` is supplied, `target_image` will be the same size as `source_image`
216     target_rows, target_cols = target_size if target_size else (source_rows, source_cols)
217     target_image = np.zeros((target_rows, target_cols), dtype=np.uint8)
218
219     # We iterate over each pixel in `target_image` and assign the appropriate intensity
220     for i_t in range(target_rows):
221         for j_t in range(target_cols):
222             # Map each target point `(i_t, j_t)` through `transformation`
223             # to obtain its corresponding source_point `(i_s, j_s)`
224             # <---
225             i_s, j_s, v = np.array([i_t, j_t, 1]) @ transformation.T
226             i_s, j_s = i_s / v, j_s / v
227             # --->
228
229             # We ignore all target points whose source points lie outside the
230             # source image. All these intensities remain 0.
231             if 0 <= i_s < source_rows - 1 and 0 <= j_s < source_cols - 1:
232                 # Assign the intensity value of target_image at `(i_t, j_t)` using the
233                 # bilinear interpolation function above.
234                 # <---
235                 target_image[i_t, j_t] = bilinear_interpolation_c(source_image, (i_s, j_s))
236                 # --->
237
238     return target_image
239
240
241 def scale_c(source_image, factor):
242     """
243     Scales the `source_image` by `factor` in both dimensions.
244
245     Args:
246         source_image (np.array): The source image
247         factor (float): Scaling factor.
248
249     Returns:
250         np.array: Scaled image
251
252     """
253     # Create a variable called `scaling` which holds a 3 x 3
254     # numpy array that corresponds to scaling by factor
255     # Note that this matrix should be T-S,
256     # so it would be the inverse of what you might think of for
257     # an S-T scaling matrix.
258     # <---
259     scaling = np.array([[1 / factor, 0, 0],
260                       [0, 1 / factor, 0],
261                       [0, 0, 1]])
262     # --->
263     return transform(source_image, scaling)
264
265 cells_scaled_c = scale_c(cells, 0.8)
266
267 if np.mean((cells_scaled_c - cells_scaled)**2) < 5:
268     print('ok', end='')
269 else:
270     print('not ok', end='')
271

```

