Programming assignment: Translate an image

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/1tSscLDV9PlmBwFEpig8i8e9YOUjyqz4-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 translation on an image. The image has already been loaded for you, in the variable named 'lena'. Your task it to translate this image by 3.75 pixels downwards and 4.3 pixels to the right.

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.

```python
# Write down the co-ordinates of the remaining three corners

# Similarly, compute 'j'
    j = int(np.floor(j_s))

# The co-ordinates of the top-left ('tl') corner are simply (i, j)
tl = i, j
```

Assignments (announcements) About the Course (preview) Ask a Question (forum) Progress (student/home) Mentor (student/mentor)
# (top-right, bottom-left, bottom-right) below.

# Use the variable names `tr`, `bl`, `br` respectively.

# <---

tr = i + 1, j + 1
bl = i + 1, j
br = i + 1, j + 1
# ---->

# Next, we compute the distance of 'source point' from top-left corner along
# vertical and horizontal directions separately.

del_i = i_s - i_t, del_j = j_s - j_t
# ---->

# Create a variable called 'pixel intensity' and assign the
# pixel value obtained by bilinear interpolating pixel values
#
# at tl, tr, bl, br.
# Use `del_i`, `del_j` computed in the previous step to obtain
# the weights for interpolation.

# <---

pixel_intensity = (1 - del_j) * (1 - del_i) * source_image[tl] + 
(1 - del_j) * del_i * source_image[tr] + 
del_i * (1 - del_j) * source_image[bl] + 
del_i * del_j * source_image[br]
# ---->

# Return np.uint8(pixel_intensity)

return np.uint8(pixel_intensity)

# """Next, we use the above function to implement a function that performs
# an arbitrary T-S transformation on a source image.

def transform(source_image, transformation, target_size=None):

    # Transforms `source_image` as dictated by `transformation`.

    Note that this function does T-S mapping. So, 'transformation' is actually from Target to Source.

    Args:

        source_image (np.array): The source image
        transformation (np.array): 3 x 3 transformation matrix
        target_size (uint, uint): Size of the target_image

    Returns:

        np.array: Transformed image

        source_rows, source_cols = image_size(source_image)
        # --->
        # When no 'target_size' is supplied, 'target_image' will be the same size as 'source_image'
        target_rows, target_cols = target_size if target_size else (source_rows, source_cols)
        target_image = np.zeros((target_rows, target_cols), dtype=np.uint8)
        # --->
        # We iterate over each pixel in 'target_image' and assign the appropriate intensity
        for i in range(target_rows):
            for j in range(target_cols):
                # --->
                # Map each target point ('i.t', 'j.t') through 'transformation'
                # to obtain its corresponding source_point ('i.s', 'j.s')
                # <---
                i_s, j_s, v = np.array([i, j, 1]) @ transformation.T
                i_s, j_s = i_s / v, j_s / v
                # <---
                # We ignore all target points whose source points lie outside the
                # source image. All these intensities remain 0.
                if 0 <= i_s < source_rows - 1 and 0 <= j_s < source.cols - 1:
                    # Assign the intensity value of target_image at (i.t, j.t) using the
                    # bilinear interpolation function above.
                    # <---
                    target_image[i.t, j.t] = bilinear_interpolation(source_image, (i.s, j.s))
                    # <---
                # return target_image

        """1. Translate 'lena_translate.png' by ($t_i = 3.75, t_j = 4.3$) pixels."""

def translate(source_image, t_i, t_j):

    # Translates the 'source_image' by 't_i' pixels along rows and by 't_j' pixels along columns.

    Args:

        source_image (np.array): The source image
        t_i (float): translation along rows
        t_j (float): translation along columns

    Returns:

        np.array: Translated image

        # Create a variable called 'translation' which holds a 3 x 3
        # numpy array that corresponds to translation of t_i along rows
        # and t_j along columns. Note that this matrix should be
        # T-S, so it would be the inverse of what you might think of for
        # an S-T translation matrix.

        # <---
        translation = np.array([[1, 0, -t_i],
                                [0, 1, -t_j],
                                [0, 0, 1]])
        # <---
        return transform(source_image, translation)

# lena = imageio.imread(image file, format='PNG')
# Call the 'translate' function above with the right parameters.
# <---
# lena_translated = translate(lena, 3.75, 4.3)
# <---

def bilinear_interpolation(source_image, source_point):

    i_s, j_s = source_point
    i = int(np.floor(i_s))
    j = int(np.floor(j_s))
    t_i = i - j
    t_j = i + 1
    bl = j + 1
    br = i + 1
    del_i, del_j = i_s - i, j_s - j

https://onlinecourses.nptel.ac.in/noc20_ee83/progassignment?name=113
pixel_intensity = (1 - del_i) * (1 - del_j) * source_image[tl] + \
(1 - del_i) * del_j * source_image[tr] + \
(del_i * (1 - del_j) * source_image[bl] + \
(del_i * del_j) * source_image[br])
return np.uint8(pixel_intensity)
def transform_c(source_image, transformation, target_size=None):
    source_rows, source_cols = image_size(source_image)
    target_rows, target_cols = target_size if target_size else (source_rows, source_cols)
    target_image = np.zeros((target_rows, target_cols), dtype=np.uint8)
    for i in range(target_rows):
        for j in range(target_cols):
            i_s, j_s, v = np.array([i, j, 1]) @ transformation.T
            i_s, j_s = i_s / v, j_s / v
            if 0 <= i_s <= source_rows - 1 and 0 <= j_s <= source_cols - 1:
                target_image[i, j] = bilinear_interpolation_c(source_image, (i_s, j_s))
return target_image
def translate_c(source_image, t_i, t_j):
    translation = np.array([[1, 0, -t_i],
                            [0, 1, -t_j],
                            [0, 0, 1]])
    return transform_c(source_image, translation)
lena_translated_c = translate_c(lena, 3.75, 4.3)
if np.mean((lena_translated - lena_translated_c)**2) < 5:
    print('ok', end=''
else:
    print('not ok', end=''

https://onlinecourses.nptel.ac.in/noc20_ee83/progassignment?name=113