hw2_interpolation

May 13, 2019

1 Part II Image Transformation

For this part you are required to write some functions for interpolation and transformation.

```
In [0]: import numpy as np
        from PIL import Image
        from matplotlib import pyplot as plt
        import math
```

Please load the example2.png.

```
In [0]: # if you are using Google Colab, please use the below codes to load image
        from google.colab import files
        from io import BytesIO
        uploaded = files.upload()
        img = Image.open(BytesIO(uploaded['example2.png']))
```

```
In [0]: # if you are using local jupyter notebook, please use the below codes to load image
        img = Image.open('example2.png')
```

```
In [0]: # change the image into a gray image
        img = img.convert('L')
        h,w = np.shape(img)
        print('height:',h,' width: ',w)
        plt.figure()
        plt.imshow(img, cmap='gray')
        plt.show()
```

1.1 Question1: Bilinear Interpolation

Here you need to implement a function for bilinear interpolation from scratch. Xq and Yq are arrays of coordinates of the points we want to interpolate.

For example, Xq=[0.5, 1.2], Yq=[0.8, 1.9] indicate that we want to interpolate the points (0.5, 0.8) and (1.2, 1.9).

The output should be a list of interpolation result.

```
In [0]: def interp2(image, Xq, Yq):
            '''
            Write your own code here.
            '''
            return interp_points
```

1.2 Question 2: Write a function that creates a 2D affine transformation matrix in homogenous and its inverse from a sequence of elementary transformations

The input is a list of operation name and its parameters. The operation is restricted to {rotation, shear, shift, scaling}.

For example, [('scaling', 1.2), ('shift', [10 20]), ('scaling', .2), ('rotation', 90)]

Your return should be the composed affine matrix, and its inverse.

```
In [0]: from numpy.linalg import inv
        import math

        def get_affine_matrix(op_list):
            '''
            Write your own code here.
            '''

            return affine_matrix, iaffine_matrix
```

1.3 Question 3: Based on the below two functions, write a code to achieve the operation of rotation and scaling.

Here you need to write a transformation function which takes the input, affine matrix, iaffine matrix and new shape of your output image. We will compare your transformation result with the functions provided by PIL after a rotation and scaling.

The return should be a 2d matrix of of the result of transforming an image.

```
In [0]: # write the code for transform function
        def transform(img, affine, iaffine, new_shape):
            '''
            Write your own code here.
            '''
```

Now we will check if your result compared with the functions from PIL. You will get full credit if you can have a similar output.

```
In [0]: # show the standard transformation result
        theta = 118
        scaling_rate = 0.5
        standard_img = (img.rotate(theta)).resize((int(h*scaling_rate), int(w*scaling_rate)))
        plt.figure()
        plt.imshow(standard_img, cmap='gray')
        plt.show()
```

```
In [0]: # show your transformation result

        # get the shape of the output
        new_shape = (np.array(np.shape(img))*scaling_rate).astype(int)
        h, w = np.shape(img)
        # get the related affine matrix
        affine, iaffine = get_affine_matrix([('shift', [-w/2, -h/2]),('rotation',theta), ('shift
        # transform the image
        transfered_img = transform(np.array(img), affine, iaffine, new_shape)
        plt.figure()
        plt.imshow(transfered_img, cmap='gray')
        plt.show()
```

2 Bonus: Write a solver function that retrieves the affine transformation (in terms of a sequence of elementary transformations) between two provided images (depicting the same object transformed by an affine transformation). Justify your approach and comment on the limitations.

```
In [0]: # generate the random transformation
        random_theta = np.random.random()*180
        random_scaling = np.random.random()*0.5 + 0.5
        random_shift_x = (np.random.random()-0.5)*w*0.1
        random_shift_y = (np.random.random()-0.5)*h*0.1

        # get the random transformed image
        random_shape = (np.array(np.shape(img))*random_scaling).astype(int)
        # affine, iaffine = get_affine_matrix([('rotation', random_theta), ('scaling', random_sc
        affine, iaffine = get_affine_matrix([('shift', [-w/2, -h/2]),('rotation',theta), ('shift
        random_transformed_img = transform(np.array(img), affine, iaffine, random_shape)

        # show the image
        plt.figure()
        plt.imshow(random_transformed_img, cmap='gray')
        plt.show()

        # show the affine matrix
        print(affine)
```

Now write your own code to get the affine matrix based on the original image and random transformed image. And give a description of your method and result.

```
In [0]: def solver(img, random_transformed_img):
            return affine_matrix
```