CO543 - Image Processing
Lab 01
E/17/219
Nawarathna K.G.I.S.

Part 1
Implement the following functions on your own using PythonOpenCV.

(a). imcomplement(I)

- Here the image is inverted.
- This can be done using the 1-I equation.

- Code :

```
[226] #implementing imcomplement
      def imcomplement(I):
        '''
          getting the inverse of an image

          I : input image
        '''
        return 1-I
```

- Results :

Input	Output
	

(b). flipud(I)

- Here the given image is flipped along the x-axis.
- This will result in an upside down image of the original image.
- Here the order of rows are changed i.e : the last row of the original image is placed at the top of the output image and the row before the last row in the input image is placed at the second row of the output image. This continues for all the rows in the input image.
- Code :

```python
#implementing flipud
def flipud(I):
  '''
    This function gets and image and output the flipped image
    flipping happens along the x-axis

    I : input image

  '''

  output = []   #this list stores the output array

  #make a list with last row of the input as the first row of the output
  for i in range(len(I)):
    currList = list(I[len(I)-i-1])
    output.append(currList)

  #convert the list to numpy array and output
  return np.array(output)
```

- Results :

Input	Output
	

(c).flip(I)

- Here the given image is flipped along the y-axis.
- This will result in a left and right switched image of the original.
- Here, for all the rows, the order of each column is changed from **start to end** to **end to start**.
- Code :

```
[237] #implementing fliplr
def fliplr(I):
  '''
    This function gets and image and output the flipped image
    flipping happens along the y-axis

    I : input image

  '''
  output = []   #this list stores the output array

  #go through all the rows
  #in each row change the order of elements
  for i in range(len(I)):
    innerList = []
    currList = list(I[i])
    size = len(currList)
    #changing the order of elements
    for j in range(size):
      innerList.append(currList[size-j-1])
    output.append(innerList)

  #convert the list to the numpy array and return it
  return np.array(output)
```

- Results :

Input	Output
	

(d).imresize(I,[x,y])

- Here the image is resized using nearest-neighbour interpolation.
- This will result in smaller or larger images of the original image.

- Code :

```
[243] def imresize(I,new_height,new_width):
        '''
          Resizing the image using nearest neighbour interpolation

          I : image to be resized
          new_height : height of the output
          new_width : width of the output

        '''

        #getting the height and the width of the current image
        old_height= len(I)
        old_width = len(I[0])

        #calculating the ratios of the rows and columns
        row_ratio, col_ratio = np.array((new_height,new_width))/np.array((old_height,old_width))

        #apply interpolation to rows
        interpolated_rows = (np.ceil(range(1, 1 + int(old_height*row_ratio))/row_ratio) - 1).astype(int)

        #apply interpolation to columns
        interpolated_columns = (np.ceil(range(1, 1 + int(old_width*col_ratio))/col_ratio) - 1).astype(int)

        #getting the combination of row and coulmn interpolations
        output = I[:, interpolated_rows][interpolated_columns, :]

        return output
```

- Results :

Input	Output
	

Part 2

Implement the 4 geometric transformation functions using OpenCV in addition to the given example.

(1). Translation

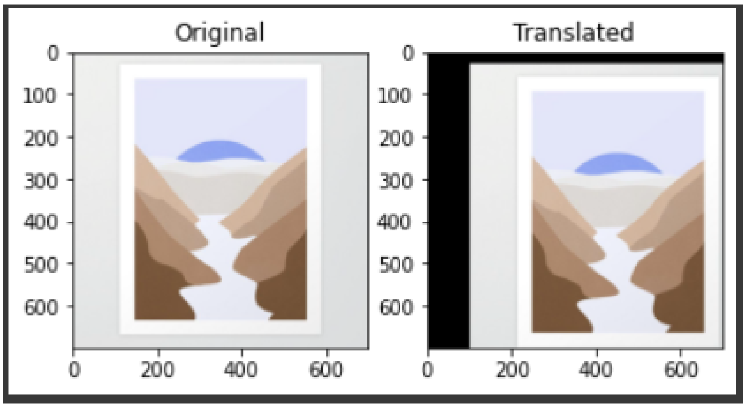
- Translation is the process of image shifting from one location to another.
- For this below matrix needs to be multiplied with the original image array.

$$M = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix}$$

- Code :

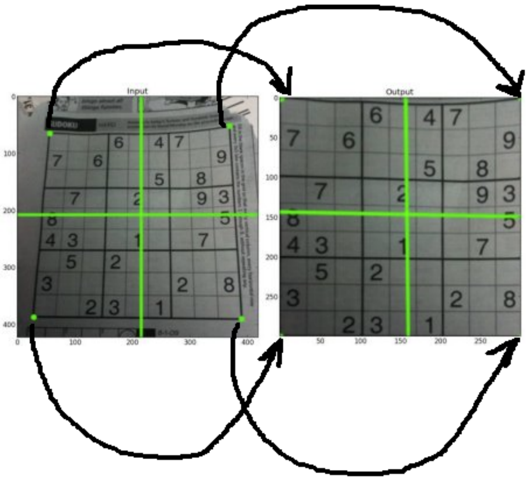
```
[247] #function definition of the translation
      def translation(I,x_shift,y_shift):

        '''
        this funtion takes an image and then shift it using the top left corner(0,0)
        to the (x_shift,y_shift).
        I : image array
        x_shift : distance along the x-axis
        y_shift : distance along the y-axis
        '''

        #get the matrix related to this transformation
        M = np.float32([[1,0,x_shift],[0,1,y_shift]])

        #get the number of rows and columns of the image
        rows,cols,ch = I.shape

        #apply the matrix to the image
        output = cv2.warpAffine(img,M,(cols,rows))

        #return the translated image
        return output
```

- Result :



(2).Projective

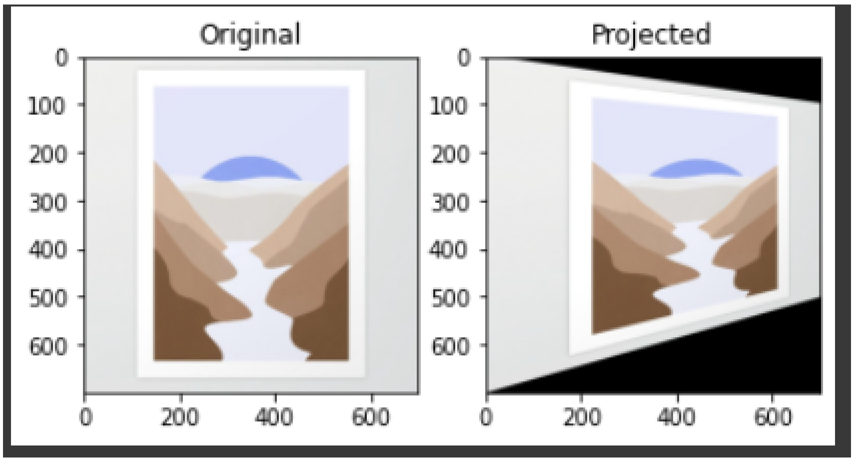
- Here projection of an image is implemented. That means we need to map given 4 points in the input image to given 4 points in the output image.



- Code :

```
[250] #function definition of the projection
      def projection(I,input_image_points,output_image_points):

        '''
          This function produces the projection of an image

          I : image array
          input_image_points : 4 points as a list of lists to indicate upper left,
          upper right, bottom left and bottom right corners in the input image.
            ex = [[56,65],[368,52],[28,387],[389,390]]

          output_image_points : 4 points as a list of lists to indicate upper left,
          upper right, bottom left and bottom right corners in the output image.
            ex = [[0,0],[300,0],[0,300],[300,300]]
        '''

        #convert list of lists to numpy arrays
        pts1 = np.float32(input_image_points)
        pts2 = np.float32(output_image_points)

        #get the matrix related to this transformation
        M = cv2.getPerspectiveTransform(pts1,pts2)

        #get the number of rows and columns of the image
        rows,cols,ch = I.shape

        #apply the matrix to the image
        output = cv2.warpPerspective(I,M,(cols,rows))

        #return the translated image
        return output
```

- Result :



(3).Euclidean

- The Euclidean image of the original image can be obtained after rotating the image by a given angle with respect to the centre.
- This can be gained by multiplying the original image by the below matrix.

$$M = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix}$$

- In the opencv documentation , it says opencv uses the below matrix so that it can rotate and scale the image at the same time.

$$\begin{bmatrix} \alpha & \beta & (1-\alpha) \cdot center.x - \beta \cdot center.y \\ -\beta & \alpha & \beta \cdot center.x + (1-\alpha) \cdot center.y \end{bmatrix}$$

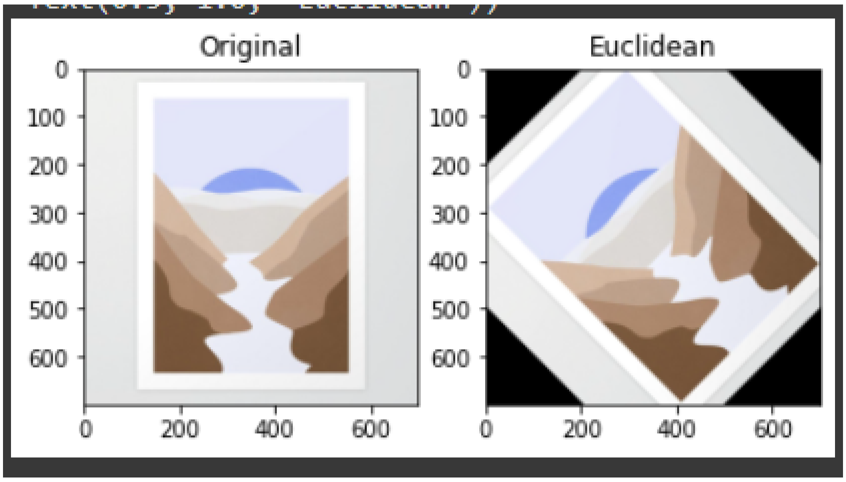
- Here ,

$$\alpha = scale \cdot \cos\theta,$$
$$\beta = scale \cdot \sin\theta$$

- Code :

```
[252] #function definition of the eulidean
      def Euclidean(I,angle):

        '''
          This function produces the euclidean of an image

          I : image array
          angle : rotation angle(clockwise) in degrees

        '''

        #get the number of rows and columns of the image
        rows,cols,ch = I.shape

        #get the matrix related to this transformation
        #rotation happens with respect to the center of the image
        #no scaling
        M = cv2.getRotationMatrix2D(((cols-1)/2.0,(rows-1)/2.0),angle,1)

        #apply the matrix to the image
        output = cv2.warpAffine(I,M,(cols,rows))

        #return the translated image
        return output
```

- Results :



  - Original image is rotated 45 degrees clockwise.

(4).Similarity

- Similarity of an image is the result when an image is rotated and resized.

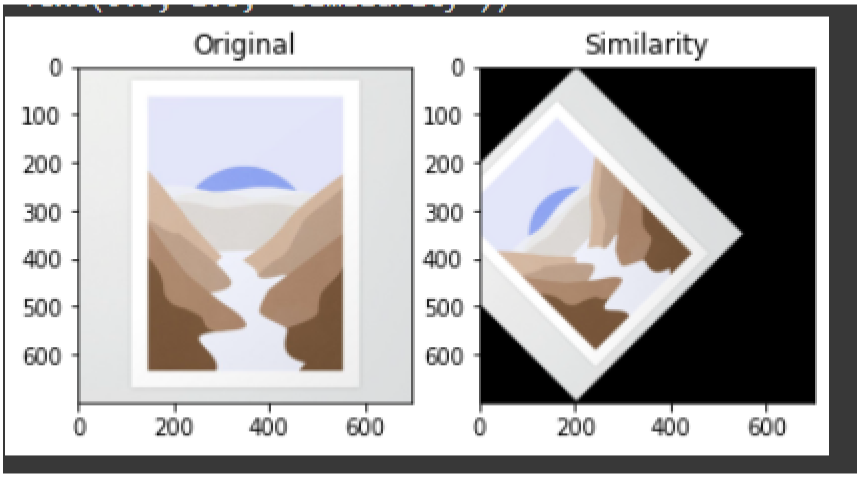
- In the code,
    1. First the resizing of the image happens.
        - For this x_scale_factor and y_scale_factor is passed.
        - Cubic interpolation is used for this.

    2. Then rotation of the image happens.

- Code :

```
[255] #function definition of similarity
      def Similarity(I,angle,x_scale_factor,y_scale_factor):

        '''
          This function produces the similarity of an image

          similarity of an image is the result when image is rotated and resized.

          I : image array
          angle : rotation angle(clockwise) in degrees
          x_scale_factor : scaling along the x-axis
          y_scale_factor : scaling along the y-axis

        '''

        #resize the image using the given parameters
        #Inter cubic interpolation method is used here
        output = cv2.resize(I,None,fx=x_scale_factor, fy=y_scale_factor, interpolation = cv2.INTER_CUBIC)

        #get the number of rows and columns of the image
        rows,cols,ch = I.shape

        #get the matrix related to this using given parameters
        M = cv2.getRotationMatrix2D(((cols-1)/2.0,(rows-1)/2.0),angle,1)

        #apply the matrix to the image
        output = cv2.warpAffine(output,M,(cols,rows))

        #return the translated image
        return output
```

- Result :



  - Here the original image is resized with the scaling factor of 0.7 along x and y axes and then rotated by 45 degrees clockwise.