Computer Vision HW1: Basic Image Manipulation

R10741015 鄭傑鴻

Sept. 7, 2022

Original Picture:

(With shape (512, 512))

Part 1:

(a) Upside down

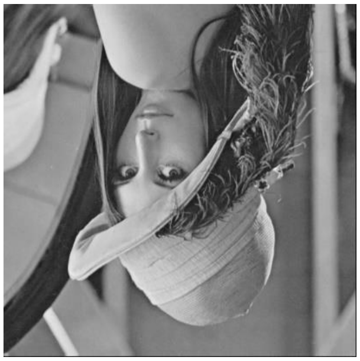
- Description:

  For every element (pixel) in the new picture, denoted as $pic\_new[i][j]$, assign pixel of the original picture $pic[511 - i][j]$ to it. (The height of the picture is 512, i & j from 0 to 511)

- Code:

```
def upsideDown(self, save=True):
    pic_new = np.zeros_like(self.pic)
    rownum = self.pic.shape[0]
    colnum = self.pic.shape[1]
    for i in range(rownum-1):
        for j in range(colnum-1):
            pic_new[i][j] = self.pic[rownum-1-i][j]
    showImg(pic_new)
    if save: cv2.imwrite('lena_upsideDown.bmp', pic_new)
```

- Resulting Image:



(b) Right-side Left

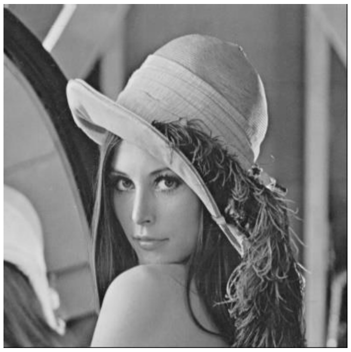
- Description:

For every element (pixel) in the new picture, denoted as $pic\_new[i][j]$, assign pixel on the original picture $pic[i][511-j]$ to it. (The width of the picture is 512, i & j from 0 to 511)

- Code:

```
def rightsideLeft(self, save=True):
    pic_new = np.zeros_like(self.pic)
    rownum = self.pic.shape[0]
    colnum = self.pic.shape[1]
    for i in range(rownum-1):
        for j in range(colnum-1):
            pic_new[i][j] = self.pic[i][colnum-1-j]
    showImg(pic_new)
    if save: cv2.imwrite('lena_rightsideLeft.bmp', pic_new)
```

- Resulting Image:



(c) Diagonally Flip

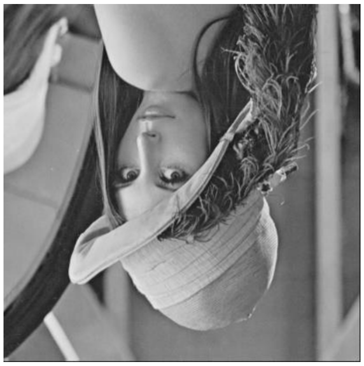
- Description:

  For every element (pixel) in the new picture, denoted as $pic\_new[i][j]$, assign pixel on the original picture $pic[511-i][511-j]$ to it. (The width and height of the picture are both 512, i & j from 0 to 511)

- Code:

```
def diagonalFilp(self, save=True):
    pic_new = np.zeros_like(self.pic)
    rownum = self.pic.shape[0]
    colnum = self.pic.shape[1]
    for i in range(rownum-1):
        for j in range(colnum-1):
            pic_new[i][j] = self.pic[rownum-1-i][colnum-1-j]
    showImg(pic_new)
    if save: cv2.imwrite('lena_diagonalFlip.bmp', pic_new)
```

- Resulting Image:



Part 2:

(d) Rotate 45 degrees clockwise

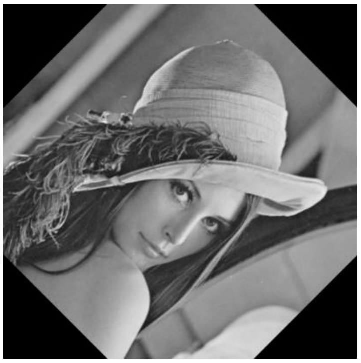
- Description:

  Since part 2 has no restrictions regarding using libraries, I use the *imutils.rotate()* method, with a negative 45 degrees counter-clockwise.

- Code:

```
def rotate45(self, save=True):
    pic_new = imutils.rotate(self.pic, -45)
    showImg(pic_new)
    if save: cv2.imwrite('lena_rotate45.bmp', pic_new)
```

- Resulting Image:



(e) Shrink height and width in half

- Description:

  Since part 2 has no restrictions regarding using libraries, I use the *cv2.resize()* method, with a new shape (256, 256).

- Code:

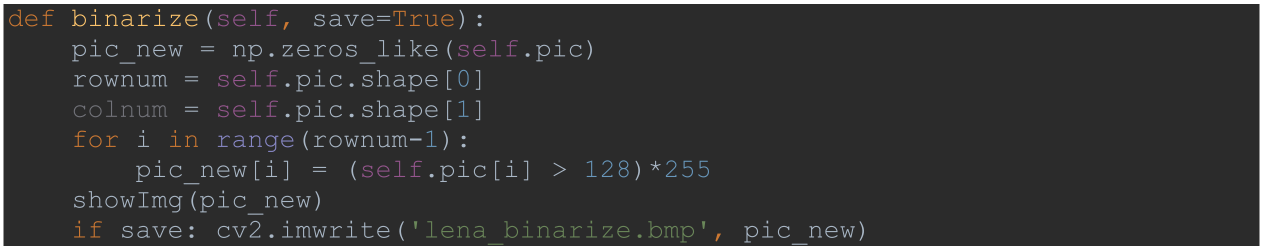
```
def shrinkHalf(self, save=True):
    h_new = int(self.pic.shape[0]/2)
    w_new = int(self.pic.shape[1]/2)
    pic_new = cv2.resize(self.pic, (h_new, w_new), cv2.INTER_AREA)
    showImg(pic_new)
    if save: cv2.imwrite('lena_shrinkHalf.bmp', pic_new)
    print('Shape for the new picture: ', pic_new.shape)
```

- Resulting Image:

```
Shape for the new picture:  (256, 256)
```

(f) Binarize at 128 to get a binary image

- Description:
  - For every element (pixel) in the new picture, denoted as $pic\_new[i][j]$, if the correspondent pixel in the original picture $pic[i][j]$ is greater than 128, assign value 255 to $pic\_new[i][j]$, else assign 0 to $pic\_new[i][j]$. (Value 255 resulting in white pixel, and value 0 result in black pixel. i & j from 0 to 511)
- Code:



```
def binarize(self, save=True):
    pic_new = np.zeros_like(self.pic)
    rownum = self.pic.shape[0]
    colnum = self.pic.shape[1]
    for i in range(rownum-1):
        pic_new[i] = (self.pic[i] > 128)*255
    showImg(pic_new)
    if save: cv2.imwrite('lena_binarize.bmp', pic_new)
```

- Resulting Image: