Image Processing HW1

姓名：林澤慶
系級：統計碩二
學號：108354015

(10%) Create a program that combines two perfectly aligned pictures (laptop_left.png and laptop_right.png). The output should be:

```
left = cv2.imread('laptop_left.png')

right = cv2.imread("laptop_right.png")
```

```
combine = np.hstack([left,right])

combine = cv2.cvtColor(combine, cv2.COLOR_BGR2RGB)
```

```
plt.title("Combines two images")
plt.axis("off")
plt.imshow(combine)
```

```
<matplotlib.image.AxesImage at 0x24de64b1490>
```

我採用numpy.hstack將兩個array進行水平合併，並透過matplotlib來顯示合併的圖片。

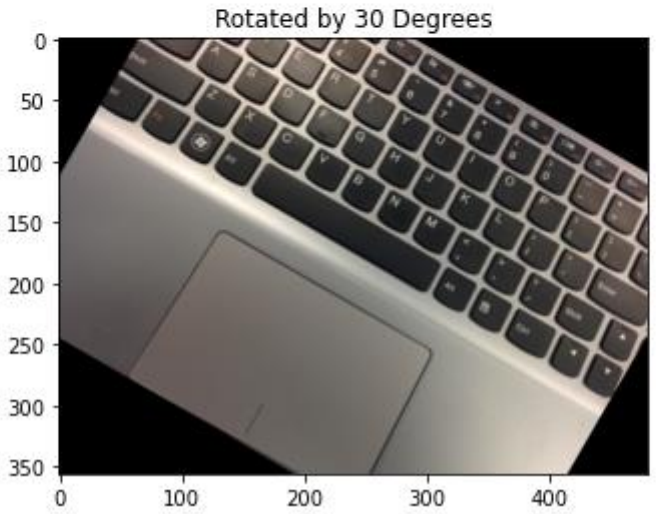
(20%) Following Q1, please rotate the combined image by 30 degrees clockwise (using an off-the-shelf function gets 10%, while implementing it by yourself gets the full credit)

```
In [5]: def rotate(image, angle, center = None, scale = 1.0):
            (h, w) = image.shape[:2]
            if center is None:
                center = (w/2,h/2)
            M = cv2.getRotationMatrix2D(center, angle, scale)
            rotated = cv2.warpAffine(image, M, (w, h))
            return rotated
```

```
In [6]: rotate_img = rotate(combine,-30)
```

```
In [7]: plt.title("Rotated by 30 Degrees")
        plt.imshow(rotate_img)
```

```
Out[7]: <matplotlib.image.AxesImage at 0x178d8cae8b0>
```



我利用opencv的getRotationMatrix2D這個function將圖像進行順時針旋轉30度，並透過matplotlib來顯示旋轉後的圖片。

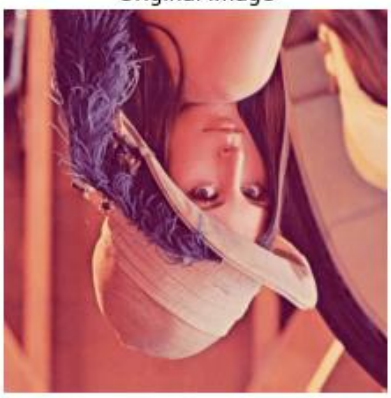
(20%) Implement a program (not using any off-the-shelf functions) to flip the image “lena_flipped.bmp.”

```
lena = cv2.imread('lena_flipped.bmp')
lena = cv2.cvtColor(lena, cv2.COLOR_BGR2RGB)
```

```
plt.title("Original image")
plt.imshow(lena)
plt.axis('off')
```

```
(-0.5, 511.5, 511.5, -0.5)
```

Original image



我先將圖片利用opencv讀取，並將圖片從原本的BGR轉換成GRB通道，透過matplotlib顯示原本的圖片。

```
def rotate_90(image):
    rotate_90_img = []
    for i in range(image.shape[0]):
        temp = []
        for j in range(image.shape[0]-1,-1,-1):
            temp.append(image[j][i])
        rotate_90_img.append(temp)
    return np.array(rotate_90_img)
```

```
lena_rotate_90 = rotate_90(lena)
lena_rotate_180 = rotate_90(lena_rotate_90)
```

```
plt.title("Flip the image")
plt.imshow(lena_rotate_180)
plt.axis("off")
plt.savefig('flip_image.jpg')
```

先定義順時針旋轉圖片90度的函數，並透過2次順時針旋轉圖片90度達成翻轉的效果，最後透過matplotlib顯示翻轉的圖片。

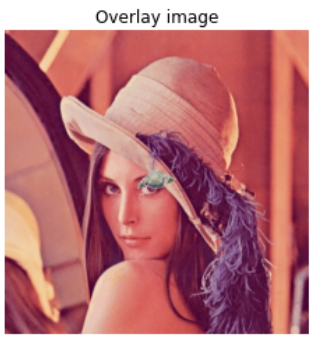
Flip the image

(25%) Please overlay the image “graveler.bmp” without the white background onto the flipped lena image.

```
graveler = cv2.imread("graveler.bmp")
graveler = cv2.cvtColor(graveler, cv2.COLOR_BGR2RGB)
```

```
class overlay():
    def __init__(self, foreground, background):
        self.foreground = foreground
        self.background = background
    def padding(self):
        self.image_padding = cv2.copyMakeBorder(self.foreground,
                (self.background.shape[0]-self.foreground.shape[0])//2,
                (self.background.shape[0]-self.foreground.shape[0]) - (self.background.shape[0]-self.foreground.shape[0])//2,
                (self.background.shape[1]-self.foreground.shape[1])//2,
                (self.background.shape[1]-self.foreground.shape[1]) - (self.background.shape[1]-self.foreground.shape[1])//2,
                cv2.BORDER_CONSTANT,
                value=[255,255,255])
    def mask(self):
        gray = cv2.cvtColor(self.image_padding, cv2.COLOR_BGR2GRAY)
        _ , self.mask = cv2.threshold(gray,250,255, cv2.THRESH_BINARY)
    def combine(self):
        self.output = np.zeros(self.background.shape)
        for i in range(3):
            self.output[:,:,i]=np.array(self.background)[:,:,i]*(self.mask/255)+self.image_padding[:,:,i]*(1-self.mask/255)
        self.output = self.output.astype('int32')
    def show(self):
        plt.title('Overlay image')
        plt.axis('off')
        plt.imshow(self.output)
```

```
Overlay = overlay(graveler,lena_rotate_180)
Overlay.padding()
Overlay.mask()
Overlay.combine()
Overlay.show()
```



我先利用opencv將graveler.bmp讀取，讓graveler設置為前景，flipped lena設置為後景，為了讓前景圖片的大小與後景圖片的大小一致，將前景透過padding方式，把前景上下左右填補白色，接續將前景轉換成灰階，並取250當作threshold形成mask，最後透過element-wise matrix multiplication的方式，利用mask將前景與後景做出合併，最後利用matplotlib呈現合併圖片的結果。

(a) Please use a watermarking technique to embed “graveler.bmp” into the flipped lena image. You need to demonstrate how to embed and retrieve “graveler.bmp” from the image with the watermark.

```
class watermark():
    def __init__(self, mark, background,coef):
        self.mark = mark
        self.background = background
        self.coef = coef
    def padding(self):
        self.image_padding = cv2.copyMakeBorder(self.mark,
                (self.background.shape[0]-self.mark.shape[0])//2,
                (self.background.shape[0]-self.mark.shape[0]) - (self.background.shape[0]-self.mark.shape[0])//2,
                (self.background.shape[1]-self.mark.shape[1])//2,
                (self.background.shape[1]-self.mark.shape[1]) - (self.background.shape[1]-self.mark.shape[1])//2,
                cv2.BORDER_CONSTANT,
                value=[255,255,255])
    def dct(self, image):
        image_float = np.float32(image)
        image_dct = np.zeros(image.shape)
        for i in range(3):
            image_dct[:,:,i] = cv2.dct(image_float[:,:,i])
        return image_dct
    def combine(self, mark_dct, background_dct, coef):
        return mark_dct  + background_dct *coef
    def extract(self, background_dct, embd_dct, coef):
        return embd_dct - background_dct * coef
    def idct(self, image):
        image_idct = np.zeros(image.shape)
        for i in range(3):
            image_idct[:,:,i] = cv2.idct(image[:,:,i])
        image_idct = np.int32(image_idct)
        return image_idct
    def show(self, image, title):
        plt.title(title)
        plt.axis("off")
        plt.imshow(image)
    def embd(self):
        ## embd image
        self.padding()
        mark_dct = self.dct(self.image_padding)
        self.background_dct = self.dct(self.background)
        mix = self.combine(mark_dct, self.background_dct, self.coef)
        mix_idct = self.idct(mix)
        self.embd_img = mix_idct//(self.coef+1)
        self.show(self.embd_img, 'embded')
        return self.embd_img
    def retrieve(self):
        ## retrieve image
        weight_embd_img = self.embd_img * (self.coef + 1)
        embd_dct = self.dct(weight_embd_img)
        extractor = self.extract(self.background_dct, embd_dct, self.coef)
        retrieve_idct = self.idct(extractor)
        self.show(retrieve_idct,'retrieve')
```

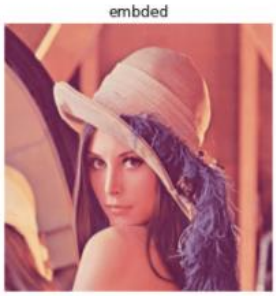
我利用dct(discrete cosine transform)的方式，將小拳石的影像嵌入至翻轉後的lena照片中。

先將lena與小拳石透過dct轉換，接著調整lena的權重與小拳相加，再透過idct投影至原本的色域空間，並將原本調整的權重調整至正常範圍，最後透過matplotlib顯示嵌入後的圖片。呈現效果如左下：

```
Watermark = watermark(graveler, lena_rotate_180,10)
```

```
embd_image = Watermark.embd()
```

embded



```
Watermark.retrieve()
```

retrieve

將嵌入後的圖片，去除原圖的資訊，提取小拳石的照片。

先將嵌入後的圖片調整權重，接著利用dct轉換與原圖轉換做相減，最後將此利用idct轉換至色域空間最後透過matplotlib顯示小拳石的圖片。呈現方式如左：

(b) Please use the JPEG standard to encode the image with the watermark using different compression ratios (at least 3 different ratios), and decode it. Please check whether you can retrieve the watermark from the decoded image using the objective quality metric, PSNR.

Image compression by bit plane

```
class bit_plane():
    def __init__(self, image, plane):
        self.image = image
        self.plane = plane
        self.row, self.col,_ = self.image.shape
    def channel(self):
        self.img_r = self.image[:, :, 0]
        self.img_g = self.image[:, :, 1]
        self.img_b = self.image[:, :, 2]
    def bit_slice_comibine(self, color):
        r = np.zeros((self.row, self.col, 8), dtype = np.uint8)

        for i in range(8):
            x = 2 ** i
            r[:,:,i] = cv2.bitwise_and(color, x)
            mask = r[:,:,i]>0
            r2 = np.copy(r)
            r2[mask] = 255

        original = r[:,:,7]

        for i in range(8,self.plane , -1):
            original = cv2.bitwise_or(original,r[:,:,i-1])
        return original
    def psnr(self, original_image, compress_image):
        mse = sum(sum(sum((original_image - compress_image)**2)))/(self.row * self.col * 3)
        psnr = 10 * np.log10((255**2)/mse)
        return psnr
    def main(self):
        self.channel()
        img_r_compress = self.bit_slice_comibine(self.img_r)
        img_g_compress = self.bit_slice_comibine(self.img_g)
        img_b_compress = self.bit_slice_comibine(self.img_b)
        img_compress = cv2.merge([img_r_compress,img_g_compress,img_b_compress])
        psnr = self.psnr(self.image, img_compress)
        plt.imshow(img_compress)
        plt.title('The bit plane compression with {} planes'.format(8-self.plane))
        plt.axis('off')
        plt.show()
        print('PSNR = {:.3f}'.format(psnr))
        return img_compress
```

將圖片每一個通道進行bit plane壓縮，來達成影像壓縮。

先將圖片每一個通道進行bit plane分解成8片，並決定要將後面多少片進行合併，最後計算壓縮圖片與原圖片之間的PSNR數值。

Retrieve the image

```
class retrieve_image():
    def __init__(self, original_image, compress_image,coef):
        self.original_image = original_image
        self.compress_image = compress_image
        self.coef = coef
    def dct(self,image):
        image_float = np.float32(image)
        image_dct = np.zeros(image.shape)
        for i in range(3):
            image_dct[:,:,i] = cv2.dct(image_float[:,:,i])
        return image_dct
    def idct(self,image):
        image_idct = np.zeros(image.shape)
        for i in range(3):
            image_idct[:,:,i] = cv2.idct(image[:,:,i])
        image_idct = np.int32(image_idct)
        return image_idct
    def retrieve(self):
        original_dct = self.dct(self.original_image)
        compress_dct = self.dct(self.compress_image)*(self.coef+1)
        extract = compress_dct - original_dct*self.coef
        retriever = self.idct(extract)
        plt.imshow(retriever+17)
        plt.axis('off')
        plt.title('retrieve')
```

利用5(a)的方法將浮水印提取出來。

以下三種在不同的壓縮比由左至右為7/8、6/8、5/8的影像處理情形。

PSNR = 51.343

PSNR = 42.779

PSNR = 35.759

上面三種不同的壓縮比皆不錯，其保留原圖的一些情形，但在提取出浮水時，在壓縮比為6/8與5/8時，浮水印有些微的雜訊，但仍可辨識出此浮水印。