Отчёт по работе с BMP-изображениями в Python-3

Задание. Написать функции, позволяющие выводить скрытые подписи изображения, поиск подписей и их проявление.

Формализуем. Программа должна находить "вотермарки" изображения (исходя из логики цвета вотермарки), и изменять яркость всех пикселов с подписью на белые либо на похожие на окружающие пикселы. После чего вывести получившееся изображение в другой графический файл.

Решение. Используя библиотеку PIL, можно решить эту задачу довольно эффективно.

Часть I

Для начала опишем логику программы, рисующую подпись (*make_mark.py*). Подключим PIL:

```
from PIL import Image, ImageDraw, ImageFont
```

Открываем изображение и накладываем на него текст с помощью стандартных функций библиотеки. Рисуем подпись примерно по центру тёмно-серым цветом. Шрифты "FreeMono.ttf" и "20443.otf" приложены к отчету в архиве.

```
img = Image.open('pic.bmp').convert("RGBA")
txt = Image.new('RGBA', img.size, (255,255,255,0))
text = "lectures.stargeo.ru"
font = ImageFont.truetype("FreeMono.ttf" , 62)
d = ImageDraw.Draw(txt)
width, height = img.size
textwidth, textheight = d.textsize(text, font)
x=width/2-textwidth/2
y=height/2-textheight
d.text((x,y), text, fill=(200,200,200,100), font=font)
```

Теперь - комбинируем изображение и текст, после чего сохраняем полученное изображение.

```
watermarked = Image.alpha_composite(img, txt)
watermarked.save('pic-mark.bmp')
```

Часть II

Опишем логику программы, стирающую подпись (*del_mark.py*), для начала подключив PIL и numpy:

```
from PIL import Image, ImageDraw, ImageFont
import numpy as np
```

Опишем две функции: select_pixel2(r,g,b,a) и handle(imgs). Первая отвечает за поиск пиксела нужного цвета и яркости, вторая - за замену на белый цвет в случае нахождения.

```
def select_pixel2(r,g,b,a):
        if r > 175 and r < 255 and g > 175 and g < 255 and b > 175 and b
< 255 and a>0:
                return True
        else:
                return False
def handle(imgs):
        for i in range(imgs.shape[0]):
                for j in range(imgs.shape[1]):
                        if select_pixel2(imgs[i][j][0],imgs[i][j][1],imgs[i][j][2],imgs[i][j][3]):
                                imgs[i][j][0] = imgs[i][j][1] = imgs[i][j][2] = 255
                                imgs[i][j][3] = 0
        return imgs
```

Теперь - откроем изображение, после чего преобразуем его в массив numpy. В массиве с помощью handle(img) найдём и замением нужные пикселы. После чего обратно преобразуем массив в картинку и сохраним ее.

```
img = Image.open('pic-mark.bmp').convert("RGBA")
img = np.array(img)
iii = handle(img)
im = Image.fromarray(np.uint8(iii))
im.save('img.bmp')
```

Часть III

Улучшаем модель. Для наглядности наложим знак другим шрифтом и будем заполнять пространство внутри бывшей скрытой надписи средним значением окружающих ее пикселов.

```
def to_border(img,i,j):
        i1,i2,i3,i4,j1,j2,j3,j4=i,i,i,i,j,j,j,j
        r=img[i][j][0]
        g=img[i][j][1]
        b=img[i][j][2]
        r0,g0,b0=0,0,0
        k,pix=img[i][j],img[i][j]
        while abs(k[0]-pix[0])<5 and abs(k[1]-pix[1])<5 and abs(k[2]-pix[2])<5
:
                i1+=1
                j1+=1
        if i1 < img.shape[0] and j1 < img.shape[0]:
                k=img[i1][j1]
```

Последнее означает, что пока цвет пиксела k не сильно отличается от pix, мы идём к границе. Таких циклов можно создать несколько, чтобы попасть на разные точки границы, после чего усреднить значение их цветов и этим значением наградить серый пиксел внутри вотермарки.

```
for count in range(5):
        if i1+count < img.shape[0] and j1+count < img.shape[1]:
                r0 += img[i1+count][j1+count][0]
                g0 += img[i1+count][j1+count][1]
                b0 += img[i1+count][j1+count][2]
        else:
                r0,g0,b0 = 255*5,255*5,255*5
        r0//=5
        g0//=5
        b0//=5
        gone = i1-i,j1-j
        return r0,g0,b0,gone
```

Функция возвращает показатели r0,g0,b0 и gone - количество шагов до границы (здесь код сильно упрощён).

Но как же применять этот обход? Если так поступать с каждым пикселом, да еще и идти в разные стороны, то получим очень больше время работы (проверено!). Поэтому логично придумать функцию с обходом серой области.

Часть IV

Опишем 2 функции: Water(a) и recurse(px,area,i,j,x,y). Первая показывает, серый (вотермарка ли) пиксел a (анналог функции select_pixel2). Вторая рекурсивно обходит область с найденным пикселом.

```
def Water(a):
    return a[0] > 175 and a[0] <= 255 and a[1] > 175 and a[1] <= 255 and a[2] > 175 and a[2] <= 255 and a[3]>=0 and a[3]<80
def recurse(px,area,i,j,x,y):
    brush=1
    if (i,j) not in area:
        if Water(px[i,j]):
            area[i,j]=px[i,j]
            for i1 in range(i-brush,i+brush):
                for j1 in range(j-brush,j+brush):
                if i1>0 and j1>0 and i1<x and j1<y:
                    recurse(px,area,i1,j1,x,y)
```

Только теперь мы будем отдельно обходить области, не заходя повторно в одну и ту же. Ниже приведен такой алгоритм. Для упрощения код немного сокращен.

```
for i in range(x):
    for j in range(y):
        px2[i,j]=px[i,j]
        flag=False
        for k in range(count):
            if (i,j) in areas[k]:
                flag=True
        if flag==True:
            continue
        else:
            if Water(px[i,j]):
                tmp=
                wave(px,tmp,i,j,x,y)
                areas[count]=tmp
                count=count+1
```

Далее приведен код для замены пикселов в серых областях на среднее значение в этой области:

```
for tmp in areas.keys():
    for tmp1 in areas[tmp].keys():
        brush1=(x*y)//(100000)
        for ii in range(tmp1[0]-brush1,tmp1[0]+brush1):
            for jj in range(tmp1[1]-brush1,tmp1[1]+brush1):
                if ii>0 and jj>0 and ii<x and jj<y:
                    px2[ii,jj]=px[ii,jj][0]//2,px[ii,jj][1]//2,px[ii,jj][2]//2
```

Часть V

Таким образом программа работает. Можно немного ее ускорить и просто заменять серые пикселы на среднее значение всех в области или на всей картинке, от чего, возможно, пострадает качество, но время работы очень порадует. Здесь приведена функция для нахождения среднего значения цвета на всей картинке. Здесь подразумевается, что мы перевели картинку в формат RGBA, то есть массив img[i, j] имеет 4 компоненты.

```
def middle(img):
        for i in range(img.shape[0]):
                for j in range(img.shape[1]):
                        a,b,c,d = img[i, j]
        return a,b,c,d
```

И напоследок напишем функцию, принимающую массив numpy и сохраняющую вотермарку в отдельный файл.

```
def save_mark(imgs):
        print("saving mark...")
        im2=Image.new("RGBA (imgs.shape[1], imgs.shape[0]), (255, 255, 255,
-100))
        im2 = np.array(im2)
        for i in range(imgs.shape[0]):
                for j in range(imgs.shape[1]):
                        if select_pixel2(imgs[i][j][0],imgs[i][j][1],imgs[i][j][2],imgs[i][j][3]):
                                im2[i][j]= imgs[i][j]
        im = Image.fromarray(np.uint8(im2))
        im.save('mark.bmp')
```