Honglu Xu
EECS 491
HW 2
Due 02/28/2018

Image Denoising

Let's make some noised image first.
A very tricky image is a black image with some bits flipped. This is very good for testing.

This is my add_noise function:

In [1]:
```
import numpy as np
from PIL import Image
import random
```

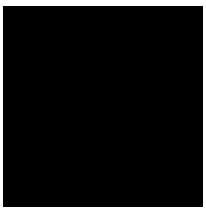
In [2]:
```
#add noise
def add_noise(image_information,p):
    image_data,image_row,image_col = image_information
    image_copy = np.copy(image_data)
    for r in range(0,image_row):
        for c in range(0,image_col):
            if random.random() < p:
                if image_copy[r][c] < 127:
                    image_copy[r][c] = 255
                else:
                    image_copy[r][c] = 0
    return (image_copy,image_row,image_col)
```

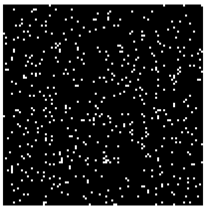
In [3]:
```
# read the image
def read_image(image_name,method):
    image = Image.open(image_name)
    image = image.convert(method)
    image_data = np.array(image)
    image_data_row = len(image_data)
    image_data_col = len(image_data[0])
    return (image_data,image_data_row,image_data_col)
```

And we can create the noised image now.

In [4]:
```
black_image_info = read_image('EECS491_hw2_p2.png','L')
black_noi_ima_info = add_noise(black_image_info,0.05)
```

In [5]:
```
img = Image.fromarray(black_noi_ima_info[0], 'L')
img.save('black_noi_ima_info.png')
```





Markov Random Field

For markov random field, it bascially says that, we have xi and yi. yi are the pixels that we observe, and xi are the pixel values that the original image should have.

1. We can somehow trust the observed pixel. (xi,yi)
   In our program, the probability to add the noise to one bit is 0.05, and we know from the commom sense that noises should be rare, if all bits are filpped, that it should not be called nosie.
2. Bits related to each other. (xi,xj)
   For a fine image, in most situation, bits that near to each other should be related to each other, so they can show a single object together.
3. And finally add bias (xi), we can create our energy function.
   $$E(X, Y) = \alpha \sum_i xi - \beta \sum_{i,j} xixj - \eta \sum_i xiyi$$
   And it will lead to:
   $$p(x, y) = \frac{1}{Z} exp(-E(x, y)$$
   Let's say we flipped a bit at k, and now we can calculate the deltaE here, and based on the demo, we can use the neighborhoods function N(x), which represents that whether x is connected to k, for it.
   $$\Delta E = -2xk(\alpha - \beta \sum_j Nj(xk) - \eta yk)$$

In [6]:
```
def convert_bin(image_information):
    image_data,image_row,image_col = image_information
    image_copy = np.copy(image_data)
    image_copy = np.int32(image_copy)
    for i in range(0,image_row):
        for j in range(0,image_col):
#             print(image_copy[i][j],"1")
            if image_copy[i][j] <= 127:
                image_copy[i][j] = -1
            else:
                image_copy[i][j] = 1
#             print(image_copy[i][j],"2")
    return (image_copy,image_row,image_col)
```

In [7]:
```
def convert_gray(image_information):
    image_data,image_row,image_col = image_information
    image_copy = np.copy(image_data)
    for i in range(0,image_row):
        for j in range(0,image_col):
            if image_copy[i][j] == 1:
                image_copy[i][j] = 255
            else:
                image_copy[i][j] = 0
    image_copy = np.uint8(image_copy)
    return (image_copy,image_row,image_col)
```

In [8]:
```python
def convert_halfhalf(image_information):
    image_data,image_row,image_col = image_information
    image_copy = np.copy(image_data)
    image_copy = np.int32(image_copy)
    for i in range(0,image_row):
        for j in range(0,image_col):
            image_copy[i][j] = image_copy[i][j]-127
    return (image_copy,image_row,image_col)
```

In [9]:
```python
def convert_halfgray(image_information):
    image_data,image_row,image_col = image_information
    image_copy = np.copy(image_data)
    for i in range(0,image_row):
        for j in range(0,image_col):
            image_copy[i][j] = image_copy[i][j]+127
            if image_copy[i][j] > 255:
                image_copy[i][j] = 255
            if image_copy[i][j] < 0:
                image_copy[i][j] = 0
    image_copy = np.uint8(image_copy)
    return (image_copy,image_row,image_col)
```

In [10]:
```python
def e_func(x,y):
    energy_matrix = np.zeros(x.shape)
    image_row = len(x)
    image_col = len(x[0])
    for i in range(0,image_row):
        for j in range(0,image_col):
            if i-1 >= 0:
                energy_matrix[i][j] += x[i-1][j]
            if j-1 >= 0:
                energy_matrix[i][j] += x[i][j-1]
            if i+1 < image_row:
                energy_matrix[i][j] += x[i+1][j]
            if j+1 < image_col:
                energy_matrix[i][j] += x[i][j+1]
    # calculate energy map
    mapE = x * (1 - 1 * energy_matrix - 1 * y)
    # get mean of matrix as energy
    E = np.mean(mapE)
    # calculate energy difference map
    dE = -2 * mapE

    return E, dE
```

In [11]:
```python
def de_noise(image_information):
    image_data,image_row,image_col = image_information
    image_copy = np.copy(image_data)
    image_filp_copy = np.copy(image_copy)
    [E,dE] = e_func(image_filp_copy,image_copy)
    i=0
    Etmp = E + 1
    while Etmp > E:
        Etmp = E
        image_filp_copy[dE < 0] *= -1
        [E, dE] = e_func(image_filp_copy, image_copy)
        i += 1
    return (image_filp_copy,image_row,image_col)
```

For the denoising part, we will go calculate all deltaE for the noised image. If the deltaE for this pixel is negative, which means this pixel is highly likely to be a noised pixel, we flip it. Then we will go through the whole image again and agin until the E is not increasing any more. Then the whole image is sort of de-noised.

I changed the method for denoising a little bit. Previously, we will use the convert function to convert the gray scale to -1 and 1. However, if we do so for a gray scaled image, the image will lost a lot of information after convertion. I tried another way, that convert all the gray scaled pixels into float numbers between the -1, 1. In that way, we can keep stall keep the relatrion between all the pixels, and after we all done, we can convert the pixels back without lossing much information. However, this is still a very tedious way to achieve. My last version is, we can still use the oringinal gray scale to do the denoise the function. Since the correction method for de-niosing is just to flip the pixels by "image_filp_copy[dE < 0] *= -1" this function, we can use change the pixels from 0~255 to -127~127. In That way, we can still flip the pixels in a gray scale.

However, there are some cons for this method. If we got a pixel that very close to 127, let's say 125. It will be -2 after convertion, and if we flip it, it will be +2. Compared to the change from -127 to 127, the change for -2 is really small. Thus, it will still be a noise piexl after we flipped it, this is not efficient.

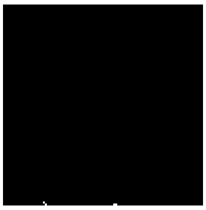
We can acctually compare those two methods, and see which one is better.

In [12]:
```
black_noi_half_info = convert_halfhalf(black_noi_ima_info)
```

In [13]:
```
image_filp_half_info = de_noise(black_noi_half_info)
```

In [14]:
```
image_flip_gray_info2 = convert_halfgray(image_filp_half_info)
```

In [15]:
```
img = Image.fromarray(image_flip_gray_info2[0], 'L')
img.save('image_flip_gray_info2.png')
```



We can see it works very well, but since this is an image that filled with black pixels, it should be very easy to denoise. Let's try something else.

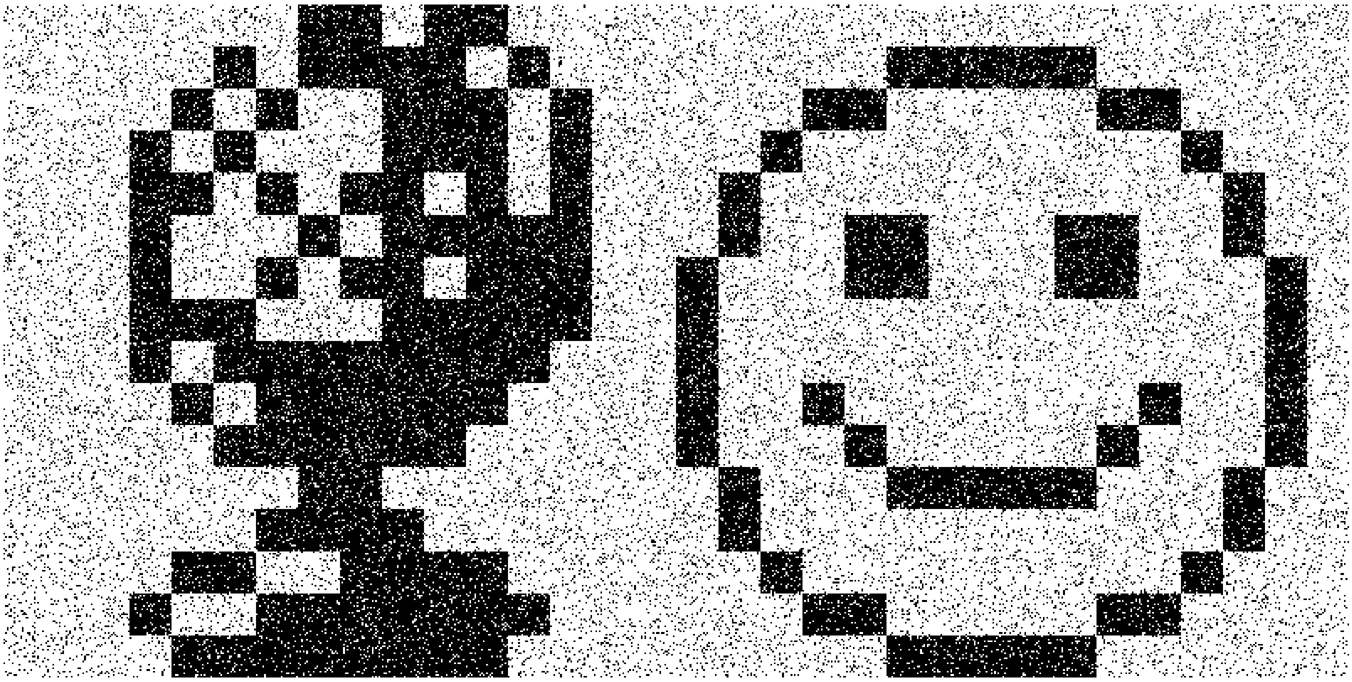
I used the happy world as in the demo.

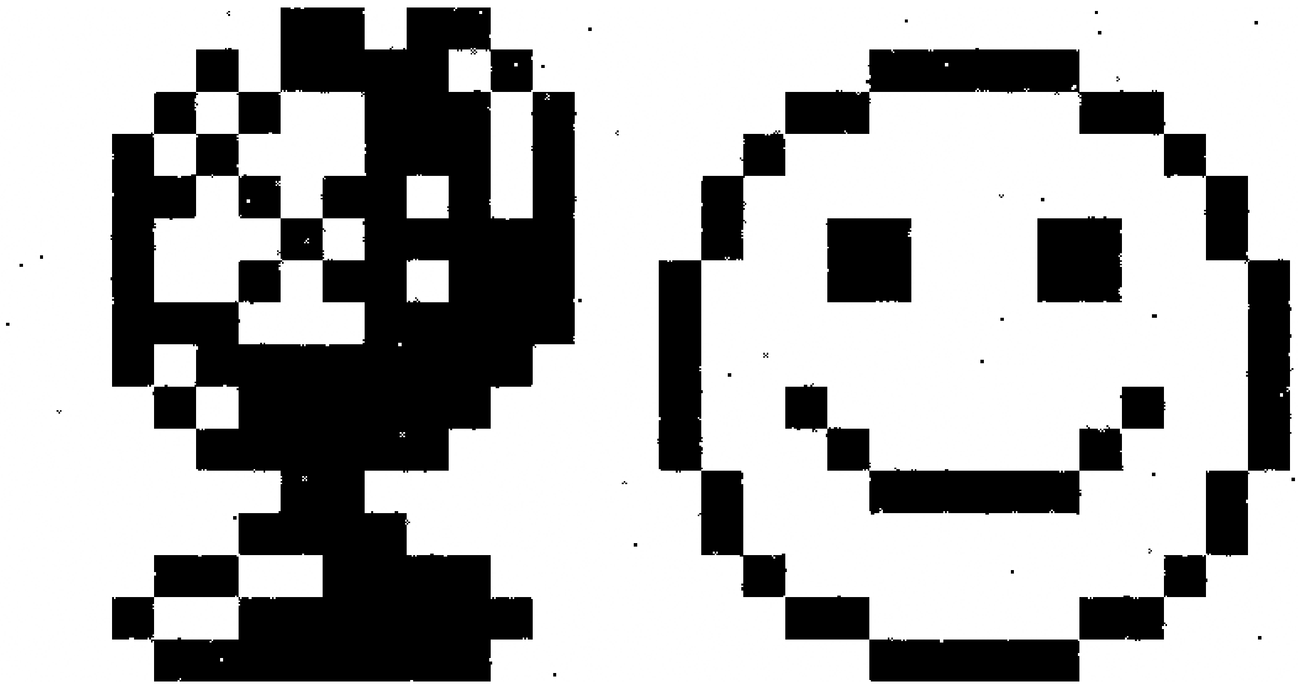
In [16]:
```
happy_image_info = read_image('happyworld_big.png','L')
happy_noi_ima_info = add_noise(happy_image_info,0.1)
```

In [17]:
```
happy_noi_half_info = convert_halfhalf(happy_noi_ima_info)
```

In [18]:
```
happy_filp_half_info = de_noise(happy_noi_half_info)
happy_filp_gray_info2 = convert_halfgray(happy_filp_half_info)
```

In [19]:
```
img = Image.fromarray(happy_filp_gray_info2[0], 'L')
img.save('happy_filp_gray_info2.png')
img = Image.fromarray(happy_noi_ima_info[0], 'L')
img.save('happy_noi_ima_info.png')
```

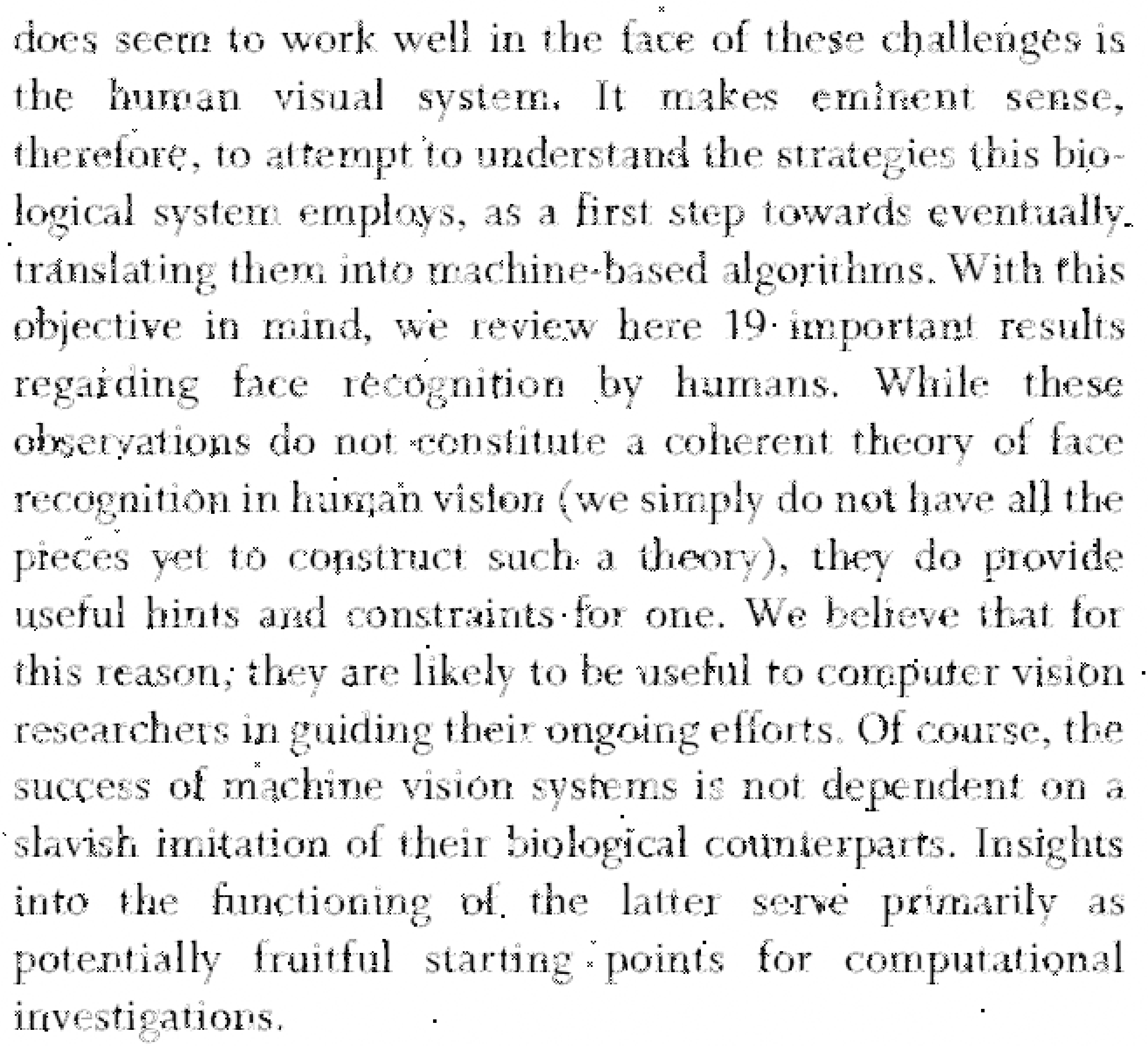




In [20]:
```
char_image_info = read_image('characters.png','L')
char_noi_ima_info = add_noise(char_image_info,0.1)
char_noi_half_info = convert_halfhalf(char_noi_ima_info)
char_filp_half_info = de_noise(char_noi_half_info)
char_filp_gray_info2 = convert_halfgray(char_filp_half_info)
```

In [21]:
```
img = Image.fromarray(char_filp_gray_info2[0], 'L')
img.save('char_filp_gray_info2.png')
img = Image.fromarray(char_noi_ima_info[0], 'L')
img.save('char_noi_ima_info.png')
```

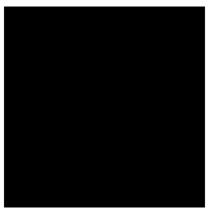
does seem to work well in the face of these challenges is the human visual system. It makes eminent sense, therefore, to attempt to understand the strategies this biological system employs, as a first step towards eventually translating them into machine-based algorithms. With this objective in mind, we review here 19 important results regarding face recognition by humans. While these observations do not constitute a coherent theory of face recognition in human vision (we simply do not have all the pieces yet to construct such a theory), they do provide useful hints and constraints for one. We believe that for this reason, they are likely to be useful to computer vision researchers in guiding their ongoing efforts. Of course, the success of machine vision systems is not dependent on a slavish imitation of their biological counterparts. Insights into the functioning of the latter serve primarily as potentially fruitful starting points for computational investigations.


does seem to work well in the face of these challenges is the human visual system. It makes eminent sense, therefore, to attempt to understand the strategies this biological system employs, as a first step towards eventually translating them into machine-based algorithms. With this objective in mind, we review here 19 important results regarding face recognition by humans. While these observations do not constitute a coherent theory of face recognition in human vision (we simply do not have all the pieces yet to construct such a theory), they do provide useful hints and constraints for one. We believe that for this reason, they are likely to be useful to computer vision researchers in guiding their ongoing efforts. Of course, the success of machine vision systems is not dependent on a slavish imitation of their biological counterparts. Insights into the functioning of the latter serve primarily as potentially fruitful starting points for computational investigations.

This is a nice de-noising result regarding to the fact that the oringinal image contains a lot of pixels that valued between 100 to 200, which we talked above that can't be de-noised efficiently.

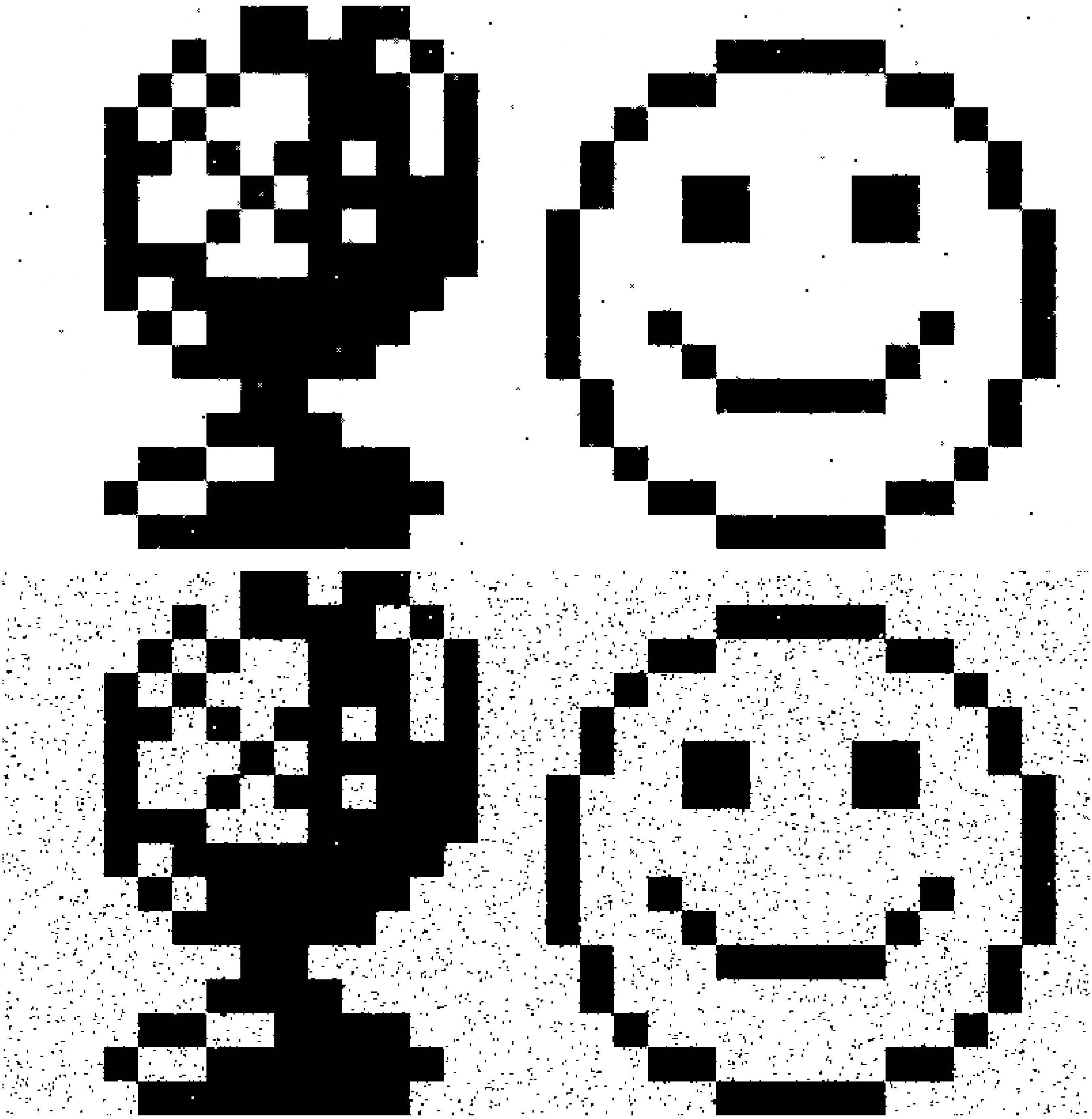
Now let's see what kind of results that the binary de-noising funciton will give.

In [22]:
```
bla_noi_bin_info = convert_bin(black_noi_ima_info)
image_filp_bin_info = de_noise(bla_noi_bin_info)
image_filp_gray_info = convert_gray(image_filp_bin_info)
img = Image.fromarray(image_filp_gray_info[0], 'L')
img.save('image_filp_gray_info.png')
```



The black test works better than the last version. The reason should be that the original image is already "binary", and of course it won't last any information after convertion.

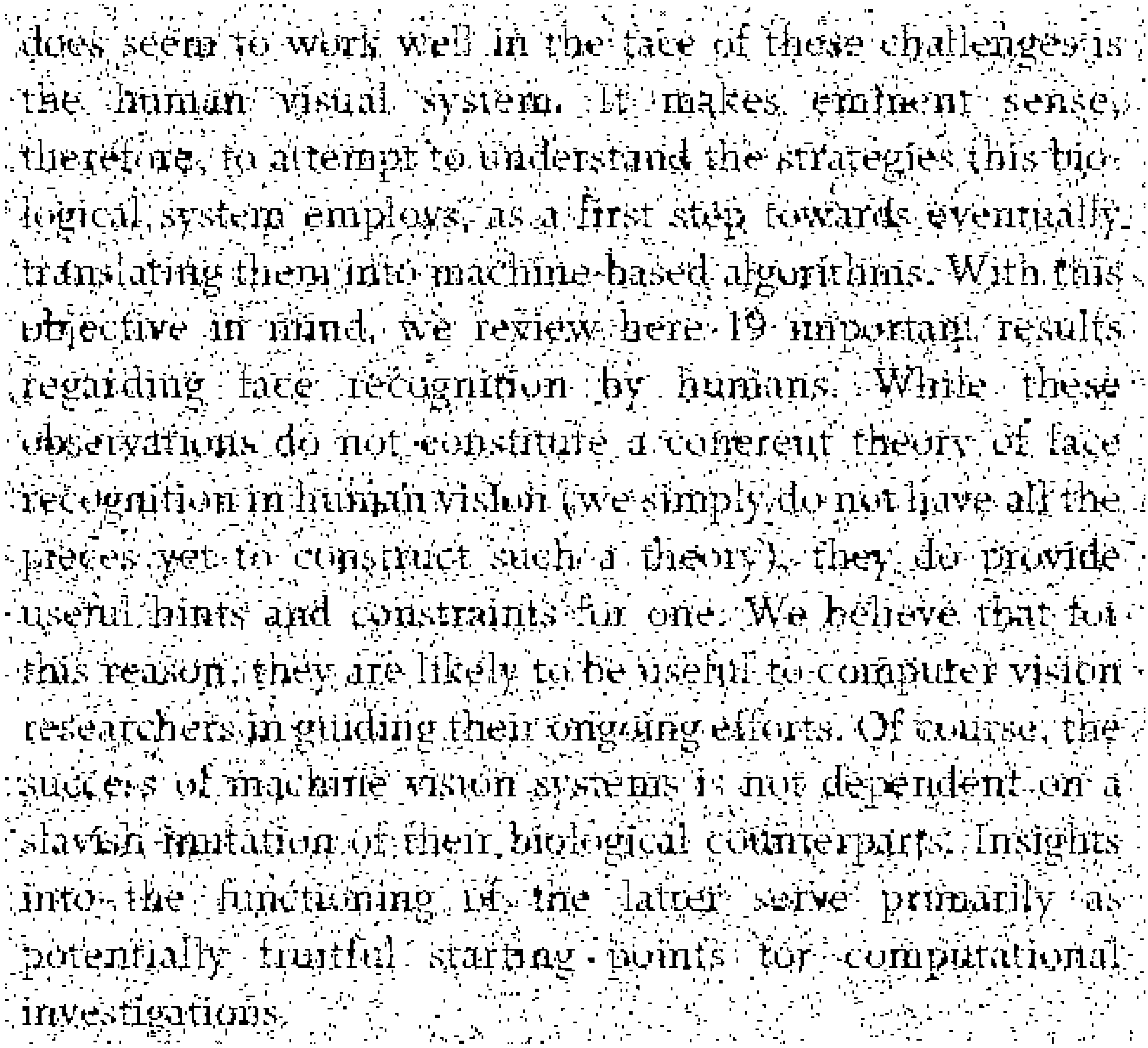
In [23]:
```
happy_noi_bin_info = convert_bin(happy_noi_ima_info)
happy_filp_bin_info = de_noise(happy_noi_bin_info)
happy_filp_gray_info = convert_gray(happy_filp_bin_info)
img = Image.fromarray(happy_filp_gray_info[0], 'L')
img.save('happy_filp_gray_info.png')
```



The upper image is from gray scaled version, and the down image is from binary version. We can tell that the gray scaled version works better. The gradient values seem to give more accuracy for the de-noise function.

In [24]:
```
char_noi_bin_info = convert_bin(char_noi_ima_info)
char_filp_bin_info = de_noise(char_noi_bin_info)
char_filp_gray_info = convert_gray(char_filp_bin_info)
img = Image.fromarray(char_filp_gray_info[0], 'L')
img.save('char_filp_gray_info.png')
```

does seem to work well in the face of these challenges is the human visual system. It makes eminent sense, therefore, to attempt to understand the strategies this biological system employs, as a first step towards eventually translating them into machine-based algorithms. With this objective in mind, we review here 19 important results regarding face recognition by humans. While these observations do not constitute a coherent theory of face recognition in human vision (we simply do not have all the pieces yet to construct such a theory), they do provide useful hints and constraints for one. We believe that for this reason, they are likely to be useful to computer vision researchers in guiding their ongoing efforts. Of course, the success of machine vision systems is not dependent on a slavish imitation of their biological counterparts. Insights into the functioning of the latter serve primarily as potentially fruitful starting points for computational investigations.


does seem to work well in the face of these challenges is the human visual system. It makes eminent sense, therefore, to attempt to understand the strategies this biological system employs, as a first step towards eventually translating them into machine-based algorithms. With this objective in mind, we review here 19 important results regarding face recognition by humans. While these observations do not constitute a coherent theory of face recognition in human vision (we simply do not have all the pieces yet to construct such a theory), they do provide useful hints and constraints for one. We believe that for this reason, they are likely to be useful to computer vision researchers in guiding their ongoing efforts. Of course, the success of machine vision systems is not dependent on a slavish imitation of their biological counterparts. Insights into the functioning of the latter serve primarily as potentially fruitful starting points for computational investigations.

Still, the upper image is from the gray scaled version, and the down iamge the from the binary version. We see that not only the de-noise function is not working very well for the binary function, the image lost a lot of information about the charaters this time, sicne the origainal image is in gray scale. Thus, it is better to use the gray scale versions.