

How to hide one picture inside another

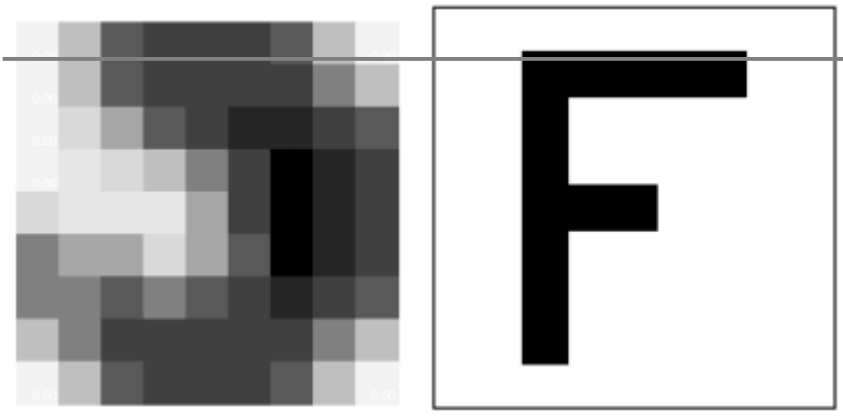
Description

default watermark

Hereâ??s an anamorphic image of Karl Marx in the aptly named Karl Marx House in Trier, Germany. Face on, the image is a blur. Side on you can see him. Thatâ??s one way of concealing an image, revealed only if you know where to stand.



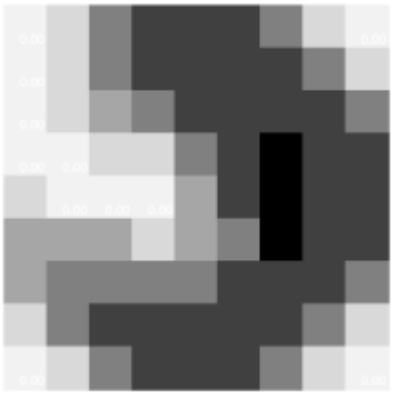
Digital images offer other methods as well. Hereâ??s a fragment of a digital image with just 10 shades of grey. Thatâ??s the host image. Next to it is another image of the same size, the letter F, that I want to hide in the host. Thatâ??s the secret image.



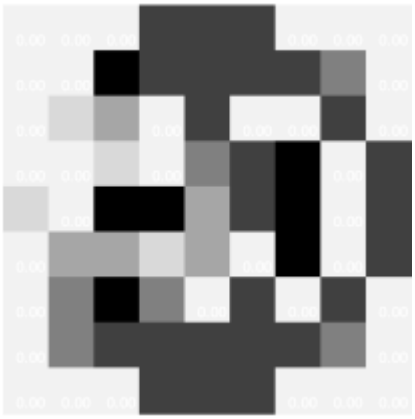
Noise tolerance

In both images, pixels that are black have a value of 10; white pixels are 0, with 9 shades in between. Pixel images are â??noise tolerant,â?• i.e. we humans are so good at identifying the patterns in a picture as a whole that we gloss over minor discrepancies in colour values. In fact, we would probably tolerate a version of the host image where the pixel values are restricted to only even numbers, i.e. 0, 2, 4, 6, 8, 10. We would recognise that the new image is of reduced quality, but still legible. Hereâ??s the same image recalculated with the reduced colour palette.

default watermark

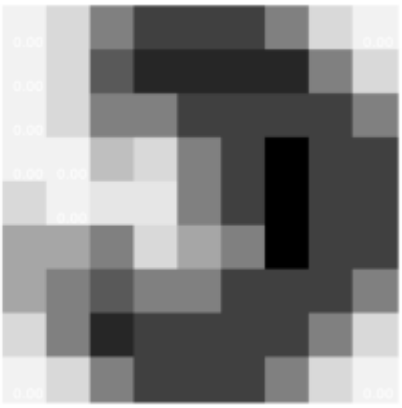


One advantage of this colour reduction is that it frees up the odd values (1, 3, 5, 7, 9) available for another image, the secret image that I want to hide. A composite image produced in that way just looks like a random amalgamation of the two. Thatâ??s not what we want.



A better algorithm to hide the secret image in the host only needs to adjust the pixel values of the original image up or down a shade to make them odd. Here, the secret image is secreted in the host following this rule.

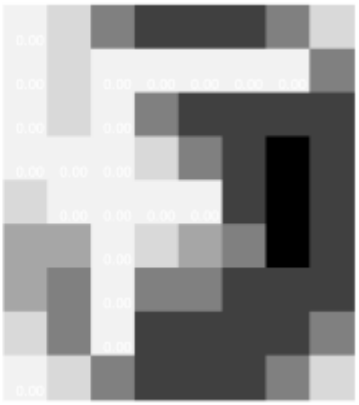
0	2	6	8	8	8	6	2	0
0	2	7	9	9	9	9	6	2
0	2	5	6	8	8	8	8	6
0	0	3	2	6	8	10	8	8
2	0	1	1	5	8	10	8	8
4	4	5	2	4	6	10	8	8
4	6	7	6	6	8	8	8	6
2	6	9	8	8	8	8	6	2
0	2	6	8	8	8	6	2	0



The secret F isnâ??t really visible unless you inspect the numerical array to the left. All the odd pixel values trace the shape of the F. The pixel values that are odd indicate the black pixels of the hidden picture. To extract the secret image, all thatâ??s needed is an algorithm that scans each pixel in turn and draws a black pixel if the value is an odd number. I have done that here. To make it more interesting I adjusted the algorithm to show the secret in white against the original host image.

default watermark



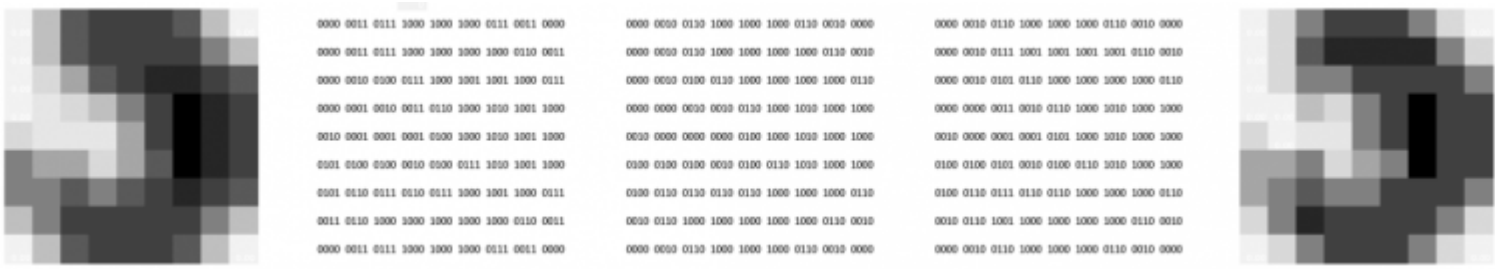


Steganography 101

The method of concealing and revealing a secret image can be scaled up to full colour with red, green and blue components for both the host image and the secret image. But rather than think in terms of odd and even pixel values, the usual method is to exploit the characteristics of binary number representations as bit strings of 0s and 1s.

The maximum pixel value in my example is 10. In binary that is 1010. The lowest value is 0, i.e. 0000 in 4 bit binary. It is the far right bit that determines that the number is odd or even. If the bit string ends in a 0 the number is even. If it ends in 1 it is odd. Another way of thinking about the calculation described above is simply to allocate the use of that last bit to the hidden image. That works here as the secret image only has 0 and 1 pixel values.

The first array of binary numbers below shows the pixel values of the host image. The second array shows the end (right) value of each bit string converted to 0, making the numbers even. The third array substitutes the bit = 1 for the end (right) value of the bit string if the secret picture is black for that pixel.



The algorithm to redraw the hidden picture only has to inspect the right end bit of the bit string for each pixel to work out what colour to draw it (black or white).

Full colour bits

default watermark

Most full colour images have at least 255 shades of red, green and blue (RGB). The binary value of 255 is 11111111. Itâ??s an 8 bit number, i.e. 2^8-1. In that case the last 3 bits could be used for the hidden image, with a loss of quality when inserted into the host image. These trailing bits are called the â??least significant bitsâ?• (LSB). You can pack extra information into those while retaining a reasonable resemblance to the original image. Adding extraneous 1s and 0s in place of the last 3 bits degrades the hostâ??s 0-255 RGB colour range to 0-31 RGB.

The hidden image would also be invisible as it provides only a subtle variation in the colour values of the original. The recovered image will be at a lower quality (colour range) than in its original state. This is explained nicely in a blog post by Kevin Salton do Prado. He shows how you would combine 8 bit RGB colour pixels. The first half of the secret bit string gets appended to the first half of the host string. The bits coloured grey here are discarded. If that were reversed, the secret would be visible and contain the host as hidden.

host
R (**1010**1101)
G (**0101**1100)
B (**0001**1010)

secret
R (**1110**0111)
G (**0001**1011)
B (**0110**0110)

combined
R (**10101110**)
G (**01010001**)
B (**00010110**)

Thatâ??s beyond the capability of my spreadsheet demonstration. In my next post I will look at why anyone would want to hide one picture inside another, and related challenges.

Reference

- Salton do Prado, Kelvin. 2018. Steganography: Hiding an image inside another. *Towards Data Science*, 18 March. Available online: https://towardsdatascience.com/steganography-hiding-an-image-inside-another-77ca66b2acb1 (accessed 18 June 2020).

Category

1. Architecture

Tags

1. cryptography
2. steganography
3. watermark

default watermark

Date Created
June 27, 2020
Author
rcoyne99